User Guide

CUSTOMER

SAP API Management, On-Premise Edition

Typographic Conventions

Type Style	Description
Example	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Textual cross-references to other documents.
Example	Emphasized words or expressions.
EXAMPLE	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
EXAMPLE	Keys on the keyboard, for example, F2 or ENTER.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Document History

Version	Date	Change
1.0	2014-08-14	Document Created
1.1	2015-01-09	Added Java Callout Policy and Node.js
1.2	2016-05-01	Updated documentation with respect to SP04 Release
1.3	2016-06-22	Updated documentation with respect to SP05 Release
1.4	2016-10-15	Updated documentation with respect to SP06 Release
1.5	2017-05-04	Updated documentation with respect to SP07 Release
1.6	2017-11-21	Updated documentation with respect to SP08 Release

Table of Contents

Data Protection and Privacy 11

1 Overview - SAP API Management 12
1.1 What is an API? 12
1.2 What is SAP API Management 12
1.3 Making services available through SAP API Management Edge 13
1.4 Creating an API product in SAP API Management Edge 14
1.4.1 Allowing a client-side app to access your API product 15
1.4.2 Steps to create API products and making them available to developers 15
1.5 Components of SAP API Management 15
1.5.1 API Platform 16
1.5.2 Analytics Services 16
1.5.3 Developer Services 17
1.6 Making APIs available through SAP API Platform 18
1.7 Developing with SAP API Platform 19
1.8 Organizations 19
1.9 Environments 23

2 Using the Management UI 24
2.1 Create a User as an Organization Administrator 24
2.2 Using the SAP API Management Platform 25
2.2.1 Understanding the Base Path 26
2.2.2 Passing Credentials 26
2.2.3 Running CURL Commands 26
2.3 Deployment Environments 28
2.4 Testing 28

3 Building APIs 29
3.1 What is an API Proxy 29
3.2 How is an API proxy represented? 30
3.3 What is a policy? 30
3.4 Understanding flows 30
3.5 Understanding routes 32
3.6 Understanding the handling of request and response data 34
3.7 API development lifecycle 37
3.7.1 Environments 37
3.7.2 Deploying API proxies to environments 38
3.7.3 Iterative development in test 39
3.7.4 Promotion to prod 39
3.8 Basic terminology 40
3.9 Best practices for API proxy design and development 41
3.9.1 Development standards 42
3.9.2 API proxy development 42
3.9.3 Message payload size 43

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.9.4 Fault Handling 43
3.9.5 Persistence 44
3.9.6 Policy and custom code 44
3.9.7 Script Callouts (Java, JavaScript, Python) 45
3.9.8 Accessing entities 46
3.9.9 Monitoring 46
3.9.10 Security 46
3.9.11 ServiceCallouts 46
3.10 Implementing API Proxies 47
3.10.1 Building a Simple API Proxy 47
3.10.2 Flows Configuration 53
3.10.3 Flow Variables and Conditions 57
3.10.4 Mapping Conditional Flows to API Resources 62
3.10.5 Introduction to Flow Variables 68
3.10.6 Reusable Shared Flows 77
3.10.7 Policy Attachment and Enforcement 81
3.10.8 Rate-Limiting 90
3.10.9 Persistence 91
3.10.10 HTTP Response Caching 93
3.10.11 Fault Handling 99
3.10.12 Pattern matching in conditional statements 113
3.10.13 Resource Files 122
3.10.14 JMS Integration 127
3.10.15 Adding CORS Support to an API proxy 131
3.10.16 Exposing a SOAP Service as an API Proxy 136
3.10.17 Streaming Requests and Responses 142
3.11 Deploying API Proxies 143
3.11.1 Deploying API proxies to Environments 143
3.11.2 Understanding Deployment 145
3.11.3 Deploying Proxies in the UI 147
3.11.4 Deploy API proxies using the Management API 151
3.12 Building Proxies with Node.js 159
3.12.1 Getting started with Node.js 160
3.12.2 Deploying a Standalone Node.js app 162
3.12.3 Understanding Support for Node.js Modules 171
3.12.4 Understanding support for the http and https modules 173
3.12.5 Understanding support for the tls module 174
3.12.6 Why the cluster module is disabled 174
3.12.7 Advanced ScriptTarget Configuration 175
3.12.8 Using the SAP API Management-access module 175
3.12.9 Debugging Node.js proxies 185
3.12.10 Understanding Support for Node.js Modules 187
3.13 Debug and Troubleshoot 191
3.13.1 Using a Trace Tool 191
3.13.2 How to use Trace 191
3.13.3 How to read a Trace 192
3.13.4 Transaction map legend 193
3.13.5 Understanding the phase details 194
3.13.6 Debugging with Trace 195
3.13.7 Selecting View Options 195

3.13.8 Downloading trace results ... 196
3.13.9 Using Offline Trace Tool ... 197
3.13.10 Troubleshooting ... 198
3.13.11 View API History ... 201
3.14 Environment Configurations ... 203
3.14.1 Creating and Editing an Environment Cache ... 203
3.14.2 Creating and Editing an Environment Key/Value Maps ... 205
3.14.3 About scope ... 206
3.14.4 Load Balancing Across Backend Servers ... 207
3.14.5 Managing target servers with the UI ... 209
3.14.6 Managing target servers with APIs ... 210
3.14.7 Configuring a TargetEndpoint to load balance across named TargetServers ... 211
3.14.8 Configuring a target server for SSL ... 215
3.14.9 Virtual Hosts ... 220

4 Secure APIs ... 226
4.1 Org Administration ... 226
4.1.1 What are organization users? ... 226
4.1.2 What SAP API Management entities do organization users work with? ... 226
4.1.3 Manage Users and Roles ... 226
4.1.4 Assign Roles ... 228
4.1.5 Creating custom roles in the UI ... 233
4.1.6 Creating roles with the API ... 235
4.2 OAuth ... 239
4.2.1 OAuth 2.0 ... 239
4.2.2 OAuth v1.0a policy ... 347
4.3 TLS/SSL ... 351
4.3.1 SSL terminology ... 351
4.3.2 Using SSL with SAP API Management ... 359
4.3.3 Configuring TLS ... 363
4.3.4 Configuring SSL ... 363
4.3.5 Configuring SSL to the Backend Service ... 372
4.3.6 KeyStores and TrustStores ... 376
4.3.7 Using SSL On Developer Portal ... 385
4.4 API Keys ... 388
4.4.1 Policy Configuration ... 388
4.4.2 Policy Attachment ... 389
4.4.3 Submitting a Request with a valid API Key ... 390
4.5 SAML ... 390
4.6 Last-Mile Security ... 391
4.6.1 Client SSL ... 391
4.6.2 Outbound Authentication ... 391
4.6.3 API key ... 392
4.6.4 OAuth Client Credentials ... 392
4.7 Content-Based Security ... 392
4.7.1 JSON threat protection ... 393
4.7.2 XML threat protection ... 393
4.7.3 General content protection ... 393
4.8 Data Masking ... 393
4.8.1 Overview ... 393

 © 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.8.2 Using Mask Configurations ... 394
4.8.3 Configuring a mask configuration resource ... 395
4.8.4 Mask Configuration API ... 396

5 Publishing APIs ... 399
5.1 Publishing Overview ... 399
5.1.1 Task 1: Create an API product ... 400
5.1.2 Task 2: Register an App Developer ... 400
5.1.3 Task 3: Register an App ... 401
5.1.4 Task 4: Document your APIs ... 402
5.1.5 SmartDocs ... 403
5.1.6 How to document your APIs by using SmartDocs ... 404
5.1.7 Configuring the SmartDocs module ... 405
5.1.8 What is an API product? ... 406
5.1.9 Creating API Products ... 407
5.1.10 Adding Developers to your Organization ... 412
5.1.11 Registering Apps ... 415
5.1.12 Using the Management API to Publish APIs ... 419

6 Analyzing APIs ... 428
6.1 Analytics Services Overview ... 428
6.1.1 Improve your API through Analytics ... 428
6.1.2 Insight that leads to action ... 428
6.1.3 What kind of data is collected and analyzed? ... 429
6.1.4 A new breed of data ... 430
6.1.5 Put Analytics up front (a best practice) ... 431
6.1.6 Your API at a glance ... 431
6.1.7 Fetch analytics data with the REST API ... 433
6.1.8 Share data with app developers ... 433
6.2 Dashboard Homepage ... 436
6.2.1 SAP API Management Dashboard ... 436
6.2.2 API Dashboards ... 436
6.2.3 Developer Dashboard ... 438
6.2.4 Exploration Dashboards ... 440
6.3 Analytics Dashboard ... 441
6.3.1 Using the analytics dashboards ... 441
6.3.2 SAP API Management Dashboard ... 445
6.3.3 Create Custom Reports ... 468
6.3.4 Use the analytics API to measure API program performance ... 472
6.3.5 Analyze API message content using custom analytics ... 483
6.3.6 Use Custom Aggregations ... 492
6.3.7 Troubleshooting with Analytics ... 495

7 References ... 498
7.1 Analytics Command Reference ... 498
7.1.1 API ... 498
7.1.2 Developers ... 500
7.1.3 Apps ... 502
7.1.4 API Products ... 504
7.1.5 API Resources ... 506

7.1.6 API program-wide analytics 508
7.2 Analytics Dimensions and Metrics 510
7.2.1 Dedicated dimensions 510
7.2.2 Computed metrics 511
7.2.3 Metrics 512
7.2.4 Functions 514
7.2.5 Dynamic Dimensions 514
7.2.6 Virtual dimensions 517
7.2.7 Report Filters 518
7.3 API Proxy Configuration Reference 519
7.3.1 API proxy structure 519
7.3.2 Configuration files and directory structure of an API proxy 520
7.3.3 Base Configuration 520
7.3.4 ProxyEndpoint 522
7.3.5 ProxyEndpoint Configuration Elements 523
7.3.6 How to Configure RouteRules 525
7.3.7 Conditional Routes 525
7.3.8 Null Routes 526
7.3.9 TargetEndpoint 526
7.3.10 TargetEndpoint Configuration 527
7.3.11 TargetEndpoint Configuration Elements 527
7.3.12 TLS/SSL TargetEndpoint Configuration 529
7.3.13 Using flow variables to set TLS/SSL values dynamically 530
7.3.14 Using references to set TLS/SSL values dynamically 531
7.3.15 TargetEndpoint with target load balancing 532
7.3.16 Policies 532
7.3.17 Policy Attachment 533
7.3.18 Flows 534
7.3.19 Conditional Flows 535
7.3.20 Flow Configuration Elements 536
7.3.21 Step processing 536
7.3.22 Resources 537
7.4 Conditions reference 537
7.4.1 Path Expressions 539
7.4.2 Operators 540
7.4.3 Relational operators and null operands 541
7.4.4 Operands 541
7.4.5 Literals 542
7.5 Endpoint properties reference 543
7.5.1 TargetEndpoint Transport Properties 543
7.5.2 TargetEndpoint Transport Property Specification 544
7.5.3 ProxyEndpoint Transport Properties 546
7.5.4 ProxyEndpoint Transport Property Specification 547
7.6 JavaScript Object Model 548
7.6.1 Background 549
7.6.2 Overview 549
7.6.3 Context 550
7.6.4 The request and response objects 550
7.6.5 The print() function 550
7.6.6 crypto object reference 551

 © 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.6.7 Context methods ... 556
7.6.8 Context Properties ... 556
7.6.9 Messages ... 557
7.7 OAuth Error Code Reference ... 563
7.7.1 Authorization Code ... 563
7.7.2 Generate AccessToken ... 564
7.7.3 Implicit ... 565
7.7.4 Refresh Token ... 566
7.7.5 Verify AccessToken ... 566
7.8 OAuth Flow Variables ... 567
7.8.1 OAuth 2.0 Flow Variables ... 567
7.8.2 OAuth 1.0a Flow Variables ... 574
7.9 Variables Reference ... 576
7.9.1 About variable scope ... 576
7.9.2 Request message variables ... 576
7.9.3 Response Message Variables ... 585
7.9.4 Message variables ... 589
7.9.5 Path Variables ... 591
7.9.6 Configuration Variables ... 593
7.9.7 System Variables ... 593
7.9.8 SSL/TLS Variables ... 595
7.9.9 Policy Variables ... 597

8 Policy Reference ... 598
8.1 Policy Reference Overview ... 598
8.2 Traffic Management Policies ... 598
8.2.1 Optimize Performance Using Cache ... 599
8.2.2 Rate Limit API Traffic Using Quota ... 629
8.2.3 Reduce latency using ResponseCache ... 637
8.3 Element reference ... 639
8.4 <ResponseCache> attributes ... 640
8.4.2 Shield APIs Using SpikeArrest ... 651
8.4.3 Reset Quota Counter Using ResetQuota ... 659
8.4.4 Throttle Backend Connections Using ConcurrentRatelimit ... 661
8.5 Security Policies ... 663
8.5.1 Authorize Requests Using OAuth 1.0a ... 663
8.5.2 Authorize Requests Using OAuth 2.0 ... 671
8.5.3 Authorize Requests using OAuthV2 Policy ... 685
8.5.4 Retrieve Token Attributes Using GetOAuthV2Info ... 716
8.5.5 Set OAuth Token Attributes Using SetOAuthV2Info ... 723
8.5.6 Delete Authorization Code or Access Token using DeleteOAuthV2Info ... 728
8.5.7 OAuth HTTP Status Codes ... 731
8.5.8 Authenticate and Authorize Using SAML 2.0 ... 737
8.5.9 Authenticate to Backend Services Using Basic Authentication ... 743
8.5.10 Manage Client Access Using IP-based Access Control ... 751
8.5.11 Minimize API Vunerabilities Using JSON Threat Protection ... 762
8.5.12 Minimize API Vulnerabilities Using XML Threat Protection ... 764
8.5.13 Search Threat Patterns Using Regular Expression ... 770
8.5.14 Verify API Keys Using API Key Validation ... 775
8.5.15 Authenticating Users and Getting User Distinguished Names in LDAP Policy ... 779

8.6 Extension Policy 785
8.6.1 Call services or APIs using ServiceCallout 785
8.6.2 Capture Analytics from API Traffic 788
8.6.3 Customize an API Using JavaScript 790
8.6.4 Customize an API Using Python 796
8.6.5 Message Logging Policy 798
8.7 Mediation Policies 806
8.7.1 ExtractVariables Policy 806
8.7.2 Apply XSL Transformations 820
8.7.3 Exception Handling With RaiseFault 823
8.7.4 Generate or Modify Messages Using AssignMessage 826
8.7.5 Persist Data Using KeyValueMap 850
8.7.6 Retrieve Entity Profiles Using AccessEntity 862
8.7.7 Validate XML and SOAP Using MessageValidation 871
8.7.8 JSON Threat Protection policy error codes 877
8.7.9 XML Threat Protection policy error codes 880
8.7.10 Convert XML to JSON 882
8.7.11 JSON payloads in Assign Message and Raise Fault 896
8.7.12 Convert JSON to XML 896
8.8 Target Servers 901
8.8.1 Create a TargetServer 901
8.8.2 Delete a TargetServer 906
8.8.3 List TargetServers in an environment 906
8.8.4 Get a TargetServer 907
8.8.5 Update a TargetServer 907
8.9 Java Callout Policy 910
8.9.1 Samples 910
8.9.2 Element reference 910
8.9.3 <JavaCallout> attributes 910
8.9.4 <DisplayName> element 911
8.9.5 <ResourceURL> element 911
8.9.6 <ClassName> element 912
8.9.7 Error reference 912
8.9.8 Referencing the Java class in a JavaCallout policy 914
8.9.9 Coding the Java class 914
8.9.10 Error handling 915
8.9.11 Specifying the default target endpoint 916
8.9.12 Testing the example 916
8.9.13 Usage notes 918
8.10 Flow Callout Policy 918
8.10.1 About the Flow Callout Policy 919
8.10.2 Element reference 919
8.10.3 <FlowCallout> attributes 919
8.10.4 <DisplayName> element 920
8.10.5 <SharedFlowBundle> element 920
8.10.6 Flow Variables 921
8.10.7 Error Codes 921

9 Glossary 922

 © 2014 SAP SE or an SAP affiliate company. All rights reserved.

Data Protection and Privacy

SAP Customer may use SAP API Management to process and monitor personal information in terms of relevant data protection legislation. It is the SAP Customer's responsibility to use the SAP API Management only in compliance with the relevant data protection laws.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

1 Overview - SAP API Management

1.1 What is an API?

An API is a technology architecture that makes it easy for one application to 'consume' capabilities or data from another application. By defining stable, simplified entry points to application logic and data, APIs enable developers to easily access and reuse application logic built by other developers. In the case of 'Web APIs', that logic and data is exposed over the network.

Since applications that consume APIs are sensitive to changes, APIs also imply a 'contract'. The contract provides some level of assurance that, over time, the API will change in a predictable manner.

SAP API Management enables you to build APIs and if you have APIs already, expose them directly, while adding a management and visibility layer. If you have HTTP enabled services, such as SOA-based Web services, they can also be exposed as APIs via SAP API Management.

1.2 What is SAP API Management

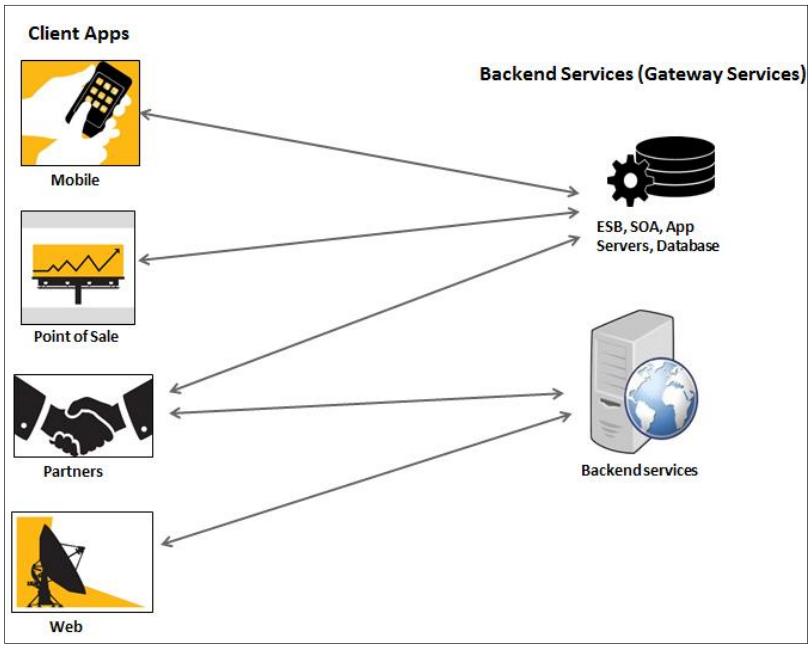
Companies today want to make their backend services available on the web so that these services can be consumed by apps running on mobile devices and desktops. A company might want to expose services that provide product pricing and availability information, sales and ordering services, order tracking services, and any other services required by client apps.

Companies often expose services as a set of HTTP endpoints. Client app developers then make HTTP requests to these endpoints. Depending on the endpoint, the service might then return data, formatted as XML or JSON, back to the client app.

The client apps that consume these services can be implemented as standalone apps for a mobile device or tablet, as HTML5 apps running in a browser, or as any other type of app that can make a request to an HTTP endpoint and consume any response data. These apps might be developed and released by the same company that exposed the services, or by third-party app developers who make use of publicly available services.

The following image shows this type of mJSOnodel:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.



Because providers make their services available over the web, they must ensure that they have taken all necessary steps to secure and protect their services from unauthorized access. As a service provider, consider:

- Security - How will you control access to your services to prevent unauthorized access?
- Compatibility - Will your services work across different platforms and devices?
- Measurability - How can you monitor your services to make sure they are available?
- And many other considerations

After a client app has been released that accesses any services, the service provider is then required to make sure that those services continue to work overtime as they add, modify, or delete those services. The service provider must also have a way to keep app developers aware of any changes to the services to ensure that client apps stay in sync with those services.

Client app developers face challenges when trying to consume services from different providers. There are many technologies available today for use by a service provider to expose its services. The same client app might have to use one mechanism to consume a service from one provider, and a different mechanism to consume a service from a different provider. App developers can even face the situation where they have to use different mechanisms to consume services from the same provider.

1.3 Making services available through SAP API Management Edge

SAP API Management Edge, which is built on Java, enables you to provide secure access to your services with a well-defined API that is consistent across all of your services, regardless of service implementation. A consistent API:

Makes it easy for app developers to consume your services.

Enables you to change the backend service implementation without affecting the public API.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Enables you to take advantage of the analytics, monetization, developer portal, and other features built into SAP API Management Edge.

The following image shows an architecture with SAP API Management Edge handling the requests from client apps to your backend services:

Rather than having app developers consume your services directly, they access an API proxy created on SAP API Management Edge. The API proxy functions as a mapping of a publicly available HTTP endpoint to your backend service. By creating an API proxy you let SAP API Management Edge handle the security and authorization tasks required to protect your services, as well as to analyze, monitor, and monetize those services.

Because app developers make HTTP requests to an API proxy, rather than directly to your services, developers do not need to know anything about the implementation of your services. All the developer needs to know is:

The URL of the API proxy endpoint.

Any query parameters, headers, or body parameters passed in a request.

Any required authentication and authorization credentials.

The format of the response, including the response data format, such as XML or JSON.

The API proxy isolates the app developer from your backend service. Therefore you are free to change the service implementation as long as the public API remains consistent. For example, you can change a database implementation, move your services to a new host, or make any other changes to the service implementation. By maintaining a consistent frontend API, existing client apps will continue to work regardless of changes on the backend.

You can use policies on the API proxy to add functionality to a service without having to make any changes to the backend service. For example, you can add policies to your proxy to perform data transformations and filtering, add security, execute conditional logic or custom code, and to perform many other actions. The important thing to remember is you implement policies on SAP API Management Edge, not on your backend server.

1.4 Creating an API product in SAP API Management Edge

An API proxy is the HTTP endpoint on SAP API Management Edge that developers use to access your backend services. While it is possible, you typically do not make individual API proxies available. Instead, you group one or more API proxies into an API product.

An API product is a bundle of API proxies combined with a service plan. That service plan can set access limits on API proxies, provide security, allow monitoring and analytics, and provide additional features. API products are also the central mechanism that SAP API Management Edge uses for authorization and access control to your APIs.

You have a great deal of flexibility when creating API products. For example, multiple API products can share the same API proxy. The following figure shows three API products. Notice that all products allow access to API proxy 3, but only product A allows access to API proxy 1.

You can set different properties on each API product. For example, you might make available one API product with a low access limit, such as 1000 requests per day, for a bargain price. You then release another API product that provides access to the same API proxy, but with a much higher access limit, for a higher price. Or, you might create a free API product that allows read-only access to your services, and then sell an API product to the same API proxies that allows read/write access.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

1.4.1 Allowing a client-side app to access your API product

When app developers decide that they want to access your services, they must first register their client app with your API product. Upon registration, an app developer receives an API key that she must then include in every request to an API proxy included in the API product. That key is authenticated and, if authentication is successful, the request is allowed to access your backend service.

At any time, you can revoke the key so that the client app no longer has access to your services. Or, you can define a time limit on a key so that the developer must refresh the key after a specific time.

You decide how to handle registration requests from developers to access your API products. By using SAP API Management Edge Developer Services, you can automate the registration process; or you can use a manual process to control access.

1.4.2 Steps to create API products and making them available to developers

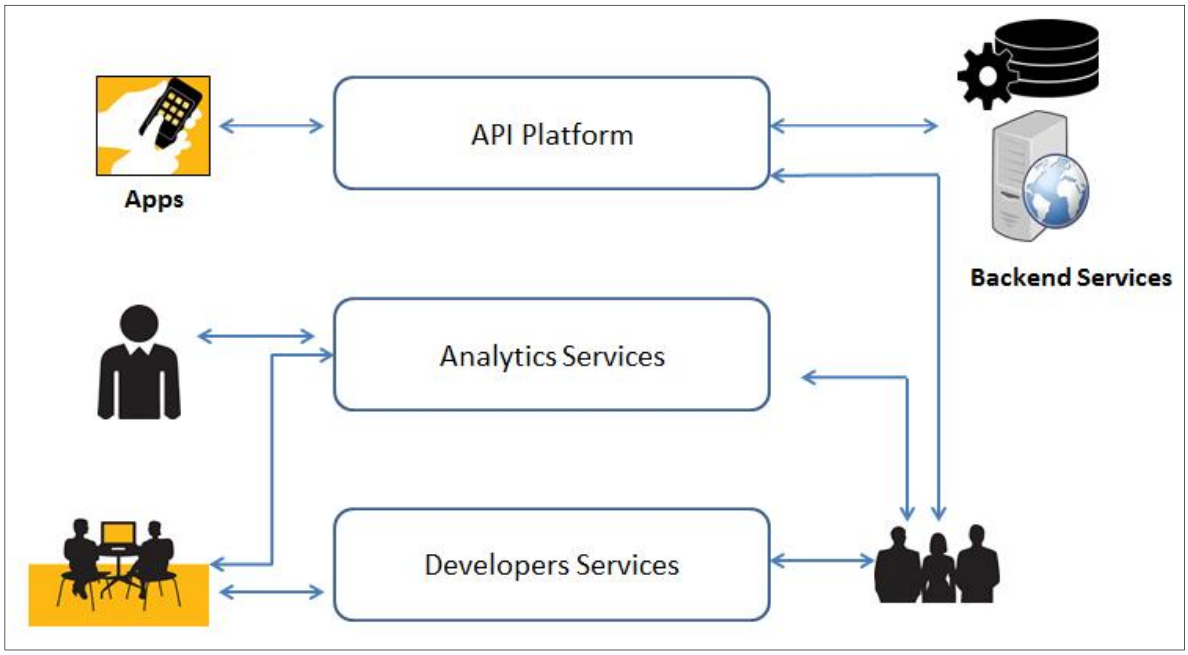
- Create one or more API proxies that map publicly available URLs to your backend services.
- Create an API product that bundles your API proxies.
- Deploy your API proxies and API product.
- Let your developers know that the API product is available.
- Once app developers know about the availability of your API product, they:
- Register their client apps with your API product.
- Receive an API key for the API product.
- Make requests to your services through API proxies (which are bundled in the API product) and pass the API key with each request.

1.5 Components of SAP API Management

SAP API Management consists of the API Platform, Analytics Services, and Developer Services that together provide a comprehensive infrastructure for API creation, security, management, and operations, as well as powerful backend services for developing client apps.

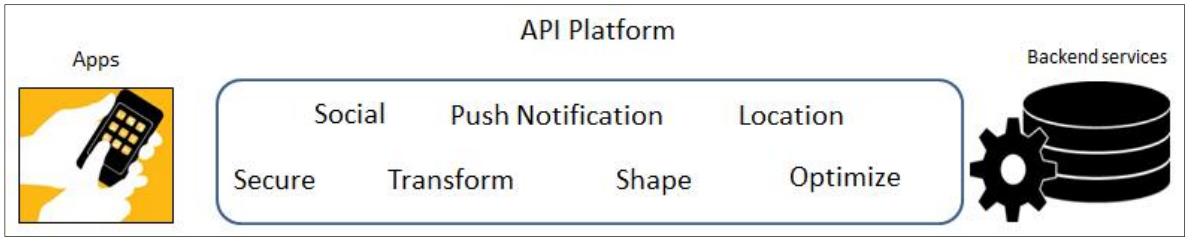
The following figure shows SAP API Management services:

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.



1.5.1 API Platform

SAP API Management API Platform are all about creating and consuming APIs, whether you're building API proxies as a service provider or using APIs, SDKs, and other convenience services as an app developer.



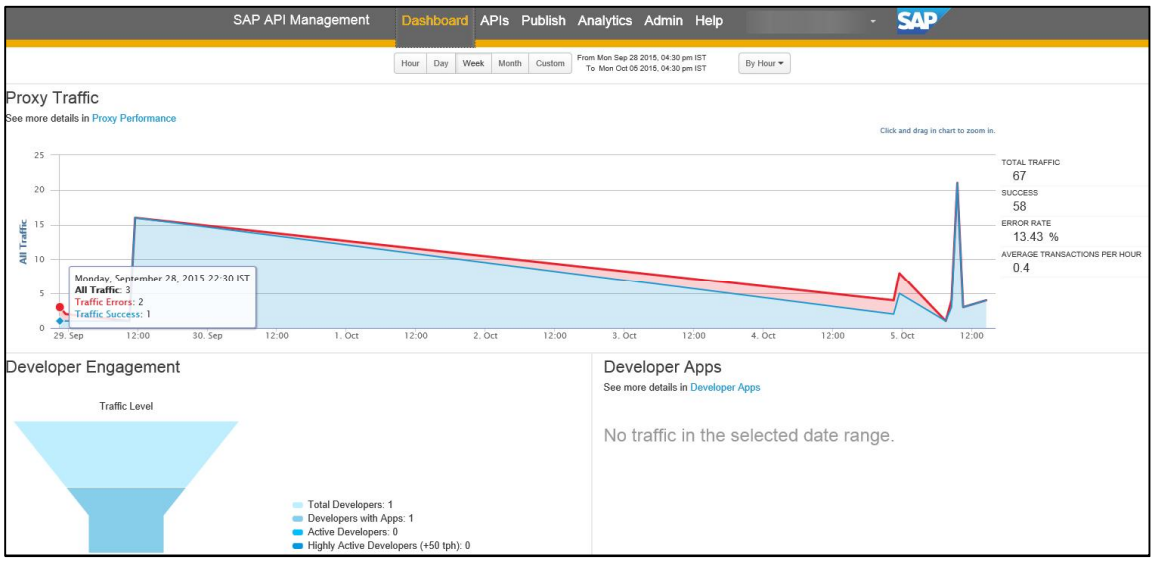
1.5.2 Analytics Services

SAP API Management Analytics Services provides powerful tools to see short and long-term usage trends of your APIs. You can segment your audience by top developers and apps, understand usage by API method to know where to invest, and create custom reports on business or operational-level information.

As data passes through SAP API Platform, several default types of information are collected including URL, IP, user ID for API call information, latency, error data, and so on. You can create policies to add other information, such as headers, query parameters, and portions of a request or response extracted from XML or JSON. This information is collected asynchronously from the actual request/response flow and therefore has no effect on API performance.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

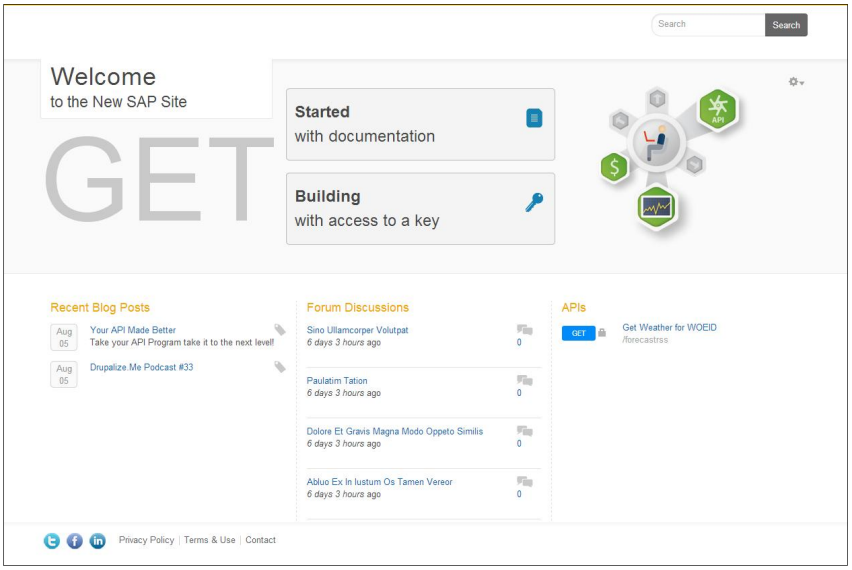
The Management UI lets you perform most analytics management tasks in a browser, as shown in the following figure:



1.5.3 Developer Services

SAP API Management Developer Services provide the tools to manage the community of app developers using your services. Developer Services offers the flexibility to work with internal and external developers and formalize these relationships with financial models.

Developer Services provides the ability to onboard developers and create a developer portal for your publicly available API products. App developers connect to your portal to access API documentation, forums, and blogs. Every SAP API Platform customer can create their own developer portal, either in the cloud or on-premise. The following figure shows the home page of the default developer portal:



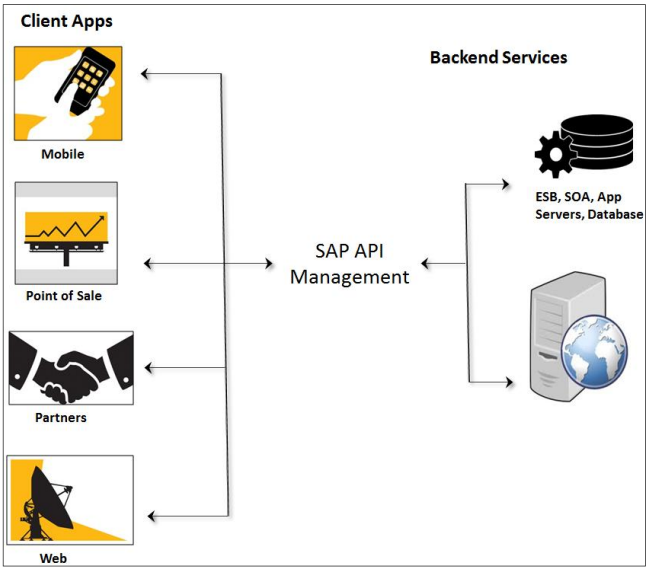
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

1.6 Making APIs available through SAP API Platform

SAP API Management, which is built on Java, enables you to provide secure access to your services with a well-defined API that is consistent across all of your services, regardless of service implementation. A consistent API:

- Makes it easy for app developers to consume your services.
- Enables you to change the backend service implementation without affecting the public API.
- Enables you to take advantage of the analytics, developer portal, and other features built into SAP API Management.

The following image shows the architecture with SAP API Management handling the requests from client apps to your backend services:



Rather than having app developers consume your services directly, they access an API proxy created on SAP API Management. The API proxy functions as a mapping of a publicly available HTTP endpoint to your backend service. By creating an API proxy you let SAP API Management handle the security and authorization tasks required to protect your services, as well as to analyze and monitor those services.

Because app developers make HTTP requests to an API proxy, rather than directly to your services, developers do not need to know anything about the implementation of your services. All the developer needs to know is:

- The URL of the API proxy endpoint.
- Any query parameters, headers, or body parameters passed in a request.
- Any required authentication and authorization credentials.
- The format of the response, including the response data format, such as XML or JSON.

The API proxy isolates the app developer from your backend service. Therefore you are free to change the service implementation as long as the public API remains consistent. For example, you can change a database implementation, move your services to a new host, or make any other changes to the service implementation. By maintaining a consistent frontend API, existing client apps will continue to work regardless of changes on the backend.

You can use policies on the API proxy to add functionality to a service without having to make any changes to the backend service. For example, you can add policies to your proxy to perform data transformations and filtering,

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

add security, execute conditional logic or custom code, and to perform many other actions. The important thing to remember is you implement policies on SAP API Management, not on your backend server.

1.7 Developing with SAP API Platform

Two different audiences make use of SAP API Management and its services:

- Service providers who create API Proxies and API products to expose backend services.
- App developers who consume these API proxies and API products to create apps for mobile and desktop devices. Client app developers also use API Platform for data persistence and management.

SAP API Platform provides tools to support both types of audiences.

As a service provider, you can use two different development methods to create, configure, and maintain API proxies and API products:

- Make HTTP requests to the RESTful API to access API Platform services.
- Use the Management UI.

1.8 Organizations

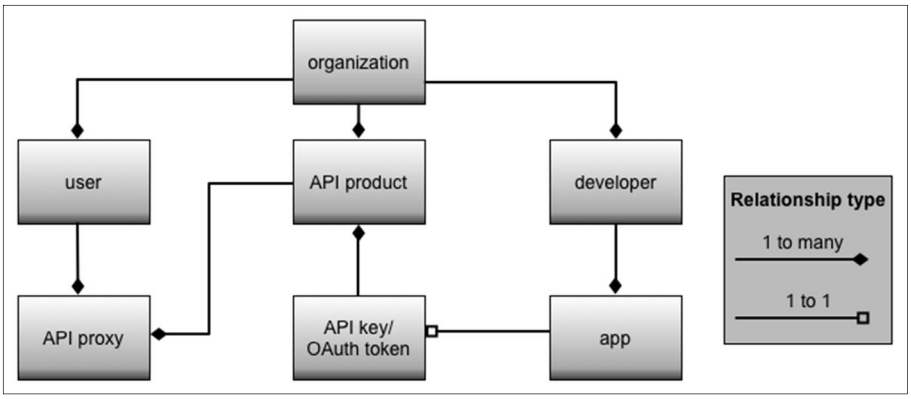
An organization represents the highest level of the App Services data hierarchy. It contains applications (and the entities and collections they contain) and is associated with one or more administrators. An organization can be representative of a company, team, or project. It allows multiple applications to be shared within the organization with other administrators.

Using the API Platform services API, you can create an organization through a form post and get an organization by UUID or name. In addition, you can activate or reactivate an organization, generate and retrieve an organization's client credentials, and get an organization's activity feed. You can also create an organization application through a form post, generate and retrieve credentials for the application, and delete the application. You can also get the applications in an organization. Additionally, you can add an admin user to an organization, get the admin users in an organization, and remove an admin user from an organization.

Before you start building APIs with SAP API Management, you should be familiar with the organizational model of SAP API Platform. This model defines how your APIs, API products, apps, and app developers are all related within SAP API Platform.

The following image shows the major components of the SAP API Platform organizational model:

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.



Example

An organization can contain one or more users and one or more API products, while an app uses a single key to access one or more API products.

Organization will be created when you perform the onboarding activity. See the Installation and Onboarding Guide for more details. Once the organization is created, you can add users to your organization, create API proxies and API products, and register developers and apps. You can later add additional organizations as necessary.

Optional: You can make the organization name as part of URL to your API proxies and part of the URL when making a request to the SAP API Management API. For example, a typical URL used to access an API proxy has the form:

`http://{org-name}-{env}.<host:port>/v1/weather/forecastrss`

The following table describes the components of the organizational model in more detail:

Component	Description
Organization	Every SAP API Management account maps to one or more organizations on SAP API Management. The organization contains a representation of all components including API proxies, API products, API packages, apps, and developers. Account holders are not limited to a single organization. Some account holders might define or be a member of multiple organizations that support different app developer communities.
User	Within an organization, where the person creating the account is automatically an administrator, you can create more users. Users make up the organization's API team, which can include people such as administrators, API proxy and API product creators, users monitoring analytics and other statistics, and any others. Different users can have different roles and access privileges. For example, define some users as Organization Administrators and Operations Administrators with privileges to modify the organization and its components. Define other users with

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Component	Description
	permissions to create API proxies and API products, but without the privileges to modify other users. Users can be members of multiple organizations. For example, your company might define multiple organizations on SAP API Management to support different developer communities. Internally though, the same people build all of the API proxies and API products and are therefore members of all of your organizations. You do not have to create an SAP API Management account—that is, create an SAP API Management organization—to be a user. An administrator can add you to an existing organization.
API Proxy	The users in an organization create one or more API proxies. An API proxy defines a mapping of a publicly available HTTP endpoint to a backend service. API proxies can also be configured to include security (such as OAuth), perform message transformation (such as XML to JSON), limit traffic to backend services, and perform other valuable operations on the request, the response, and with service callouts. SAP API Platform collects data for analytics on API proxies.
API Product	The users in an organization create one or more API products, where an API product is a bundle of API proxies combined with a service plan. That service plan can set access limits on API proxies, provide security, allow monitoring and analytics, and provide additional features. SAP API Platform collects data for analytics on API products.
Developer	An organization contains one or more developers who build the apps that consume the APIs (assembled into API products) defined by your organization. Developers consume APIs but cannot create APIs or perform any other actions in the organization. Developers can be internal to your company, they can be partners, or they can be external developers who pay for access to your APIs. Developers must be registered in your organization before they can register an app and receive an API key to access your APIs. As an API provider, it is up to you to determine how to add, update, or remove developers in your organization. You can manually add them through the SAP API Platform management UI, create a developer portal to register them through a website, or define your own registration mechanism by using the SAP API Platform management API.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Component	Description
	A developer is not required to have an account on SAP API Platform, and most developers will not need to know anything about SAP API Platform. If the developer does have an account on SAP API Platform, it is typically as a user in a different organization, or to use the API Platform. Because your API developers are registered in your organization, you can use SAP API Platform to monitor and collect analytic information on developers and their use of your APIs.
App	Developers create one or more client apps that consume your APIs. Developers must register their apps with your organization. An App in SAP API Platform is a representation of a developer's actual app that provides the developer with an API key to pass in with every request to your APIs. Because all apps are registered in your organization, you can use SAP API Platform to monitor and collect analytic information on the app and on its use of your APIs.
API key / OAuth token	Depending on the authorization mechanism you define for your APIs, the app passes an API key along with every request to your APIs. If that key is valid, the request is allowed. SAP API Platform supports different types of authentication, such as a simple API key, two-legged OAuth, three-legged OAuth, and others. As an API provider, you must define a way for developers to register their apps. It is by registering their app that you return to the developer the key required to access your APIs. At the time of app registration, the developer can choose to access a single API product or multiple API products. The developer's actual app uses the same key to access all API products associated with the app (the registered representation of the developer's app in SAP API Platform). At any time, you can revoke the key so that the developer's app no longer has access to your APIs (even though the registered representation of the developer's app still exists in your organization). Or, you can define a time limit on a key so that the developer must refresh the key after a specific time.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

1.9 Environments

Every organization has a unique software development lifecycle (SDLC). It is often necessary to synchronize and align API proxy deployment with the same processes that you use today for developing, testing, and deploying other applications.

API Platform provides tools and RESTful APIs that enable you to integrate API proxy deployment and management into your organization's SDLC. A common usage of the RESTful API is to write scripts or code that programmatically deploy API proxies, or that migrate API proxies from one environment to another, as part of a larger automated process that also deploys or migrates other applications. API Platform makes no assumptions about your SDLC (or anyone else's, for that matter). Rather, it exposes atomic functions that can be coordinated by your development team to automate and optimize your API development lifecycle.

Every organization on SAP API Management has at least two deployment environments that are available for API proxies: 'test' and 'prod'. The distinction between the two environments is arbitrary — each environment is simply identified by a different set of network addresses (URLs). The goal is to provide you with a domain in which you can build and verify API proxies before the API is exposed to external developers.

You can leverage these environments to synchronize API proxy development processed with your SDLC. Each environment is defined by a network address, enabling you to segregate traffic between the API proxies that you are working on, and those that are being accessed by apps at runtime. The network addresses available for each environment is defined in the set of VirtualHosts available in that environment.

Inbound server SSL is automatically enabled for each environment. The default VirtualHost is pre-defined in each environment during onboarding. The secure VirtualHost can be created. Default defines an HTTP address, while secure defines an HTTP/S address, with pre-configured server-side SSL. In an API proxy configuration, you indicate which VirtualHosts the ProxyEndpoint should listen on. When promoting to prod, you typically disable HTTP by removing the default VirtualHost from the API proxy configuration. See the Installation and Onboarding Guide for more information.

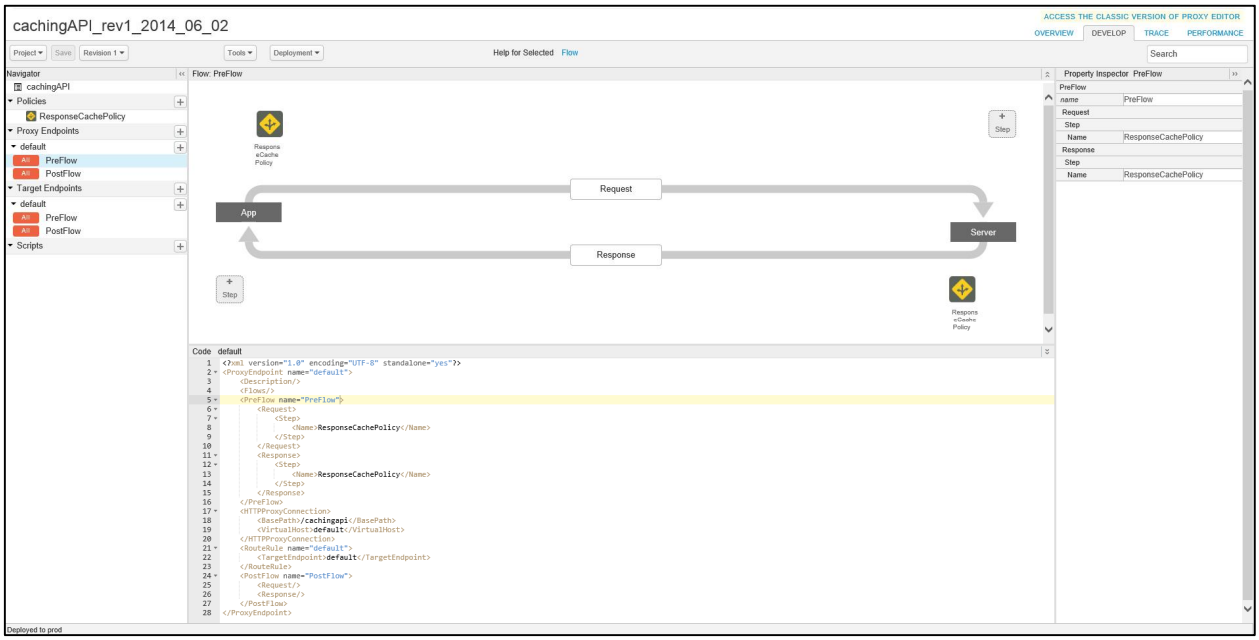
Environments also provide segregation of data and resources. You can, for example, set up different caches in test and prod, which can be accessed only by API proxies executing in that environment. Additionally, API keys that are issued in the test environment are not valid in the prod environment, and vice-versa.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

2 Using the Management UI

The UI provides browser-based tooling that lets you perform most of the tasks necessary to create, configure, and manage API proxies and API products. There are still a few tasks that require you to use the RESTful APIs, though.

The following image shows the Management UI that you can use to create and configure an API proxy:



The UI simplifies many interactions with platform in the context of a browser-based tool. In the UI, you can:

- Create API proxies
- Create API products
- Manage developers, apps, and other parts of your organization
- Configure your test and production environments
- Access the Console
- Access App Services
- Implement JavaScript

2.1 Create a User as an Organization Administrator

An organization administrator can add a new user on SAP API Management Platform. The new user is automatically added to the administrator's organization.

When the administrator creates new user, you are automatically added to the administrator's organization with the role determined by the administrator. This is typically the way you will be added as a new user to an existing organization.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

To add a new user account:

1. Log in to SAP API Management Platform with an account that has administration privileges for the organization.

   If you are a member of multiple organizations, select the correct organization from the list in the upper right of the SAP API Management UI.
2. Choose the Admin tab.
3. Choose + User. The "New User" screen appears.
4. Enter the user's Email address, First Name, and Last Name.
5. Select the user's Role. See Manage users and roles for more information. Set the Role as either:
   - User
   - Business User
   - Organization Administrator
   - Operations Administrator
6. Choose Save.
   The new user will receive an email with further instructions.

To add an existing user to your organization

Use this same procedure to add an existing user to your organization. The only difference is that in step 4, if you enter the email address of an existing user, the First Name and Last Name fields are automatically filled.

To delete a user from your organization

To delete a user from your organization, select the user in the Organization Users table and choose Delete. This only removes the user from the current account. If the user is a member of multiple accounts, they remain in the system.

2.2 Using the SAP API Management Platform

The SAP API Management Platform is a RESTful API that enables you to access SAP API Management using any HTTP client. The API enables programmatic scripting to perform development tasks, such as deployment of API proxies. The RESTful API also provides access to low-level capabilities that are not exposed by the management UI for reasons of usability. The API is designed to be lightweight and to impose few restrictions on the client code that consumes it.

As a RESTful API, the SAP API Management Platform follows a pattern in which capabilities are exposed as API 'resources'. API 'resource' is a term derived from the web application description language (WADL) specification. According to the WADL specification, a RESTful application can be described as a set of resources.

Each resource, in turn, defines a set of methods that act on it. Some examples of the resources exposed by the SAP API Management API are API proxies, API products, apps, and developers.

Following RESTful principles, you can call HTTP GET, POST, PUT, and DELETE methods on any of the API resources.

- GET: Retrieves a list of resources, or the profile of a specific resource.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- POST: Creates a resource, or passed in a parameter, performed an action on resource. For example, when you revoke OAuth access tokens, you use the POST method along with the parameter action=revoke.
- PUT: Updates an existing resource.
- DELETE: Deletes an existing resource.

2.2.1 Understanding the Base Path

Your organization name is used in the base path for the requests to the SAP API Management Platform API. You can locate your organization name under User Settings in the SAP API Management UI.

To retrieve a list of API proxies in your organization, you would call GET on:

```
http://<host:port>/v1/organizations/apibuilders/apis
```

Many resources are segregated according to environments. Two environments are provided by default: test and prod. An example of a resource that is scoped by environment is Cache. A default Cache, called 'mycache' is set up in every environment. You can list Caches by calling GET on the Cache resource as follows:

```
https://<host:port>/v1/organizations/apibuilders/environments/test/caches
http://<host:port>/v1/organizations/apibuilders/environments/prod
```

2.2.2 Passing Credentials

The API enforces HTTP Basic Authentication. You always need to pass in your username and password for your account. These can be passed in as a Base64 encoded header, or as parameters (as demonstrated below) in an HTTP client. The following is an example of an HTTP Basic Authentication header:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

2.2.3 Running CURL Commands

To consume the SAP API Management API you require an HTTP client. Many examples in the documentation provide sample API requests using cURL, a widely-used HTTP client. If you need to install cURL, you can download it from http://curl.haxx.se.

If you cut and paste a cURL command from the doc to your terminal, you must replace the following variables in the command with information for your SAP API Management account:

- `email`: The email for your account
- `password`: The password for your account in an account
- `myorg`: The name of your organization
- `host:port` : The address of the system where the SAP API Management API Platform is installed.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Note

On Windows, make sure to download a version of cURL that includes the libcurl library. Also on Windows, you might need to use a flag to disable the trust check that cURL performs against the SSL certificate presented by the SAP API Management. You can do this by adding -k to each request you submit via cURL on the command line. This only disables the trust check and does not disable encryption.

Example

```
$ curl -k http://<host:port>/v1/organizations/{org_name}/apis -u myname:mypass
```

If you are using the SAP API Management samples, set the values for your organization, username, and password in the file `/setup/setenv.sh`. Once you have done so, you can run the simplified deploy and invoke scripts provided for each sample API proxy.

Note

Calls to the management API support gzip compression on responses. If you set 'Accept-Encoding: gzip, deflate' in your API calls, any response greater than 1024 bytes gets returned in gzip format.

Example: making an API call to SAP API Management

In this example, you make an API call to SAP API Platform. The API call in this example returns a list of the names of all API proxies in your organization. Copy and paste the following command into the terminal window on your computer.

```
$ curl http://<host:port>/v1/organizations/{org_name}/apis -u myname:mypass
```

For brevity, you can abbreviate /organizations as /o. For example, if your username is me@mycompany.com, your password is secret, and your account in the organization called apifactory, then the request looks like this:

```
$ curl http://<host:port>/v1/o/apifactory/apis -u me@mycompany.com:secret
```

The response should contain a JSON array with the API proxies which are created:

```
[ "oauth", "weatherapi" ]
```

cURL has a few tools that can make you life easier. For example, you often need to see the HTTP headers associated with a request. To obtain this detail about an HTTP request, you can use the -v flag provided by curl. For example:

```
$ curl http://<host:port>/v1/o/{org_name}/apis -u myname:mypass -v
```

If you only want to see the HTTP headers on the response from the SAP API Management (and not the content), then you can use the -I flag.

```
$ curl http://<host:port>/v1/o/{org_name}/apis -u myname:mypass -I
```

Example: returning XML

JSON is the default format for response messages. If you require XML, you can add an HTTP Accept header to get an XML response instead of JSON:

```
$ curl -H "Accept:text/xml" http://<host:port>/v1/organizations/apifactory/apis -u
me@mycompany.com:secret
```

When POSTing or PUTting payloads in XML, use the Content-type HTTP header:

```
$ curl -H "Content-type:text/xml" -X POST -d \
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
'<XMLPayload>
 </XMLPayload> ' \
http://<host:port>/v1/organizations/apifactory/apis \
-u me@mycompany.com:secret
```

2.3 Deployment Environments

Every organization using SAP API Management automatically has at least two environments they can use to develop, test, and deploy APIs: 'test' and 'prod'. Use the test environment to develop and test your APIs before making them publicly available. Only your internal developers can access APIs deployed to the test environment. Deploy your APIs to the prod environment to make them publicly available to app developers.

2.4 Testing

SAP API Platform provides a trace tool that lets you debug end-to-end request and response flows. The trace results display request and response headers and payloads, policy execution, variable values, and any errors that may have occurred during the flow.

Key data points for use in troubleshooting:

- Timestamps: Use timestamps to see how long each step takes to execute. Comparing timestamps helps you isolate the policies that are taking the longest to execute that are slowing down your API calls.
- Base path: By verifying the base path, you can ensure that a policy is routing the message to correct server.
- Results of policy execution: These results let you see if the message is being altered as expected, such as if the message is being transformed from XML to JSON, or if the message is being cached.

Each Trace session is broken down into the following major steps:

- Original request received from client: Displays the verb and URI path of the request from the client app, headers, body data, and query parameters.
- Request sent to your backend service: Displays the request message sent to the backend service by the API proxy.
- Response returned by the backend service: Displays the response headers and payload returned by the backend service.
- Final response sent to client: The response message returned to the requesting client app once the response flow has executed.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3 Building APIs

3.1 What is an API Proxy

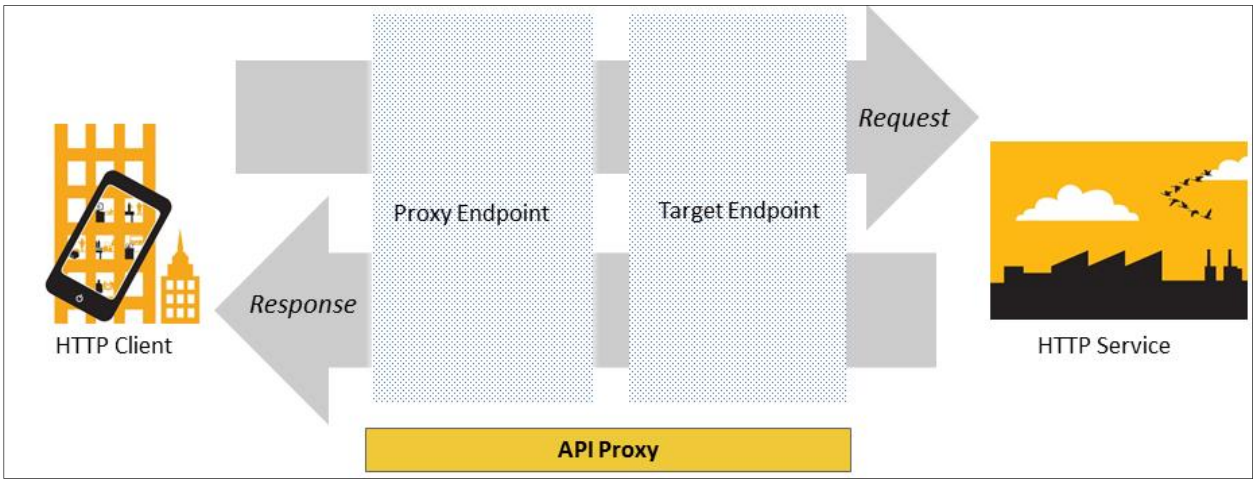
You expose APIs on SAP API Management by implementing API proxies. An API proxy is a bundle of XML configuration files and code (such as JavaScript) that implements the facade for your backend HTTP services. API proxies decouple the developer-facing API from your backend services, shielding developers from code changes and enabling you to innovate at the SAP API Management without impacting internal applications and development teams. As development teams make backend changes, developers continue to call the same API without any interruption.

API proxies manage request and response messages using a 'pipeline' processing model that defines 'Flows'. To customize the behavior of your API, you attach Policies to request and response Flows.

In an API proxy configuration, there are two types of endpoints:

- ProxyEndpoint: This configuration manages interactions with apps that consume your API. You configure the ProxyEndpoint to define the URL of your API. You usually attach Policies to the ProxyEndpoint to enforce security, quota checks, and other types of access control and rate-limiting.
- TargetEndpoint: This configuration manages interactions with your backend services on behalf of consumer apps. You configure the TargetEndpoint to forward request messages to the proper backend service. You usually attach Policies to the TargetEndpoint to ensure that response messages are properly formatted for the app that made the initial request.

You can visualize API proxies as shown by the graphic below. A basic request and response exchange between an app (HTTP client) and a backend service is managed in an API proxy by a ProxyEndpoint and TargetEndpoint.



CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.2 How is an API proxy represented?

An API proxy consists of a bundle of XML configuration files and code (such as JavaScript and Java). SAP API Management provides several ways for you to create API proxies, including:

- Using the SAP API Management UI to define an API proxy in a Graphical User Interface (GUI).
- Creating XML files, along with any other supporting files, that define your API proxy and then importing them into SAP API Management.
- Using the SAP API Management's API to create your API proxies by making a series of REST requests to SAP API Management.

3.3 What is a policy?

SAP API Management enables you to control API behavior without writing any code by using policies. A policy is like a module that implements a specific, limited management function as part of the proxy request/response flow. Policies are designed to let you add common types of management capabilities to an API easily and reliably. Policies provide features like security, rate-limiting, transformation, and mediation capabilities, saving you from having to code and maintain this functionality on your own.

3.4 Understanding flows

API proxies manage request and response messages using a pipeline processing mode, where the pipeline consists of a sequence of flows. You attach SAP API Management policies to different flows in the pipeline, depending on the type of policy and the action that you want to perform.

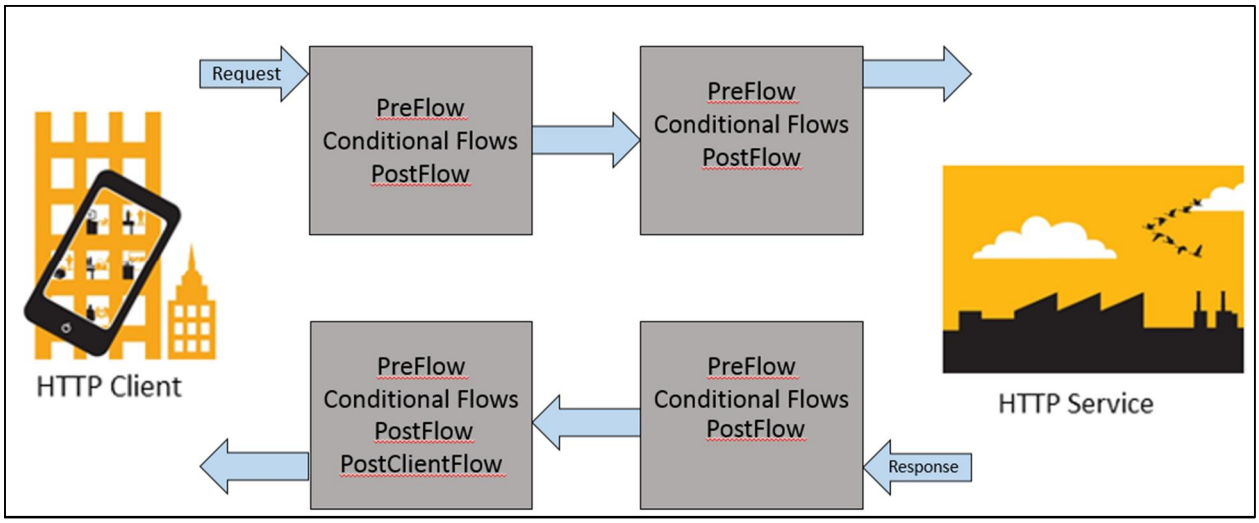
What are flows?

API proxies define request and response flows that are executed in a specific order. The order allows you to apply logic and behavior at specific points in the API proxy execution. The request and response flows are subdivided into proxy and target segments. Each segment is subdivided into the following flow 'stages':

- PreFlow: Always executes before any conditional flows.
- Conditional flows: One or more flows, each of which has an associated condition. Conditional flows tell SAP API Management, "When you see this, perform this logic." For example, when an API call is a GET on the /movies resource (the condition), transform the response to JSON (through a policy attached to the conditional flow). Only one flow executes per transaction—the first flow whose condition evaluates to true.
- PostFlow: Always executes after any conditional flows.
- PostClientFlow: An optional flow that executes after the response message has been sent to the requesting client app. (This is a specialized flow, and only MessageLogging policies can be attached to it.)

The following figure shows a basic request and response exchange between an app and a backend service. In this figure, the ProxyEndpoint and TargetEndpoint have been expanded to show the flows used to process the request and response:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.



This may look complicated, but it's fairly simple once you understand a few use cases. Flows are executed in the following order in both the Proxy Endpoint and the Target endpoint, first in the request, then in the response.

1. PreFlow

PreFlows always execute first, and the logic you attach in the PreFlow applies to all calls made to the API proxy. They are useful when you need to make sure that a policy or code executes before anything else happens. For example, you usually don't want an API proxy to waste any resources on an unauthenticated user. Also, you don't want to service an app that has exceeded its quota. To support these requirements, you put security and quota policies in the PreFlow segment. That way, you don't need to worry about a condition failing to evaluate. The policies will always execute before any other processing takes place.

2. Conditional Flow

API proxy programming starts to get interesting when you implement 'branching' logic for an API. For example, you might want to convert XML to JSON only when the requesting app is running on a mobile device; or create a new HTTP response header when the /foo API resource is called; or you might want to return a targeted ad based on the data in the user's request. You can do this by setting up conditional flows.

3. PostFlow

As with PreFlow, the logic you attach to the PostFlow applies to all calls made to the API proxy. PostFlow is useful when you need to log some data, send a notification that something happened, transform the message format, and so on.

4. PostClientFlow (response only)

You can add an optional PostClientFlow to the ProxyEndpoint that executes after the request is returned to the requesting client app. Only MessageLogging policies can be attached to this flow, which are executed after the response is sent. This reduces API proxy latency and makes information available for logging that is not calculated

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

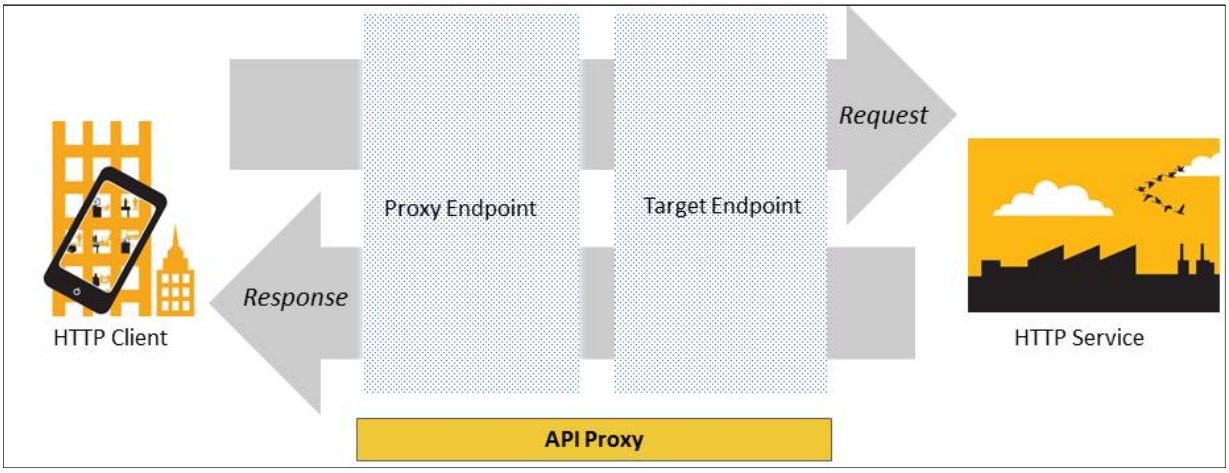
until after the response is sent, such as the client.send.start.time andclient.send.end.time. The flow is used primarily for measuring the time interval between the start and end timestamps for the response message.

3.5 Understanding routes

A route determines the path of a request from the ProxyEndpoint to the TargetEndpoint. Included in the route is the URL used to access the API ProxyEndpoint and the URL of the backend service defined by the TargetEndpoint.

Determining the URL of the API proxy endpoint

The following image shows a request coming in to the ProxyEndpoint from an app, and that request being directed to the backend service:



After you create an API proxy on SAP API Management, the default URL that an app uses to access the proxy has the form:

```
http://{org-name}-{env-name}.<host:port>/{base-path}/{resource-path}
https://{org-name}-{env-name}.<host:port>/{base-path}/{resource-path}
```

where:

`{org-name}` is your organization name. This name is created when you create an account on SAP API Management.

`{env-name}` is the SAP API Management environment name. By default, all SAP API Management organizations created in the cloud are provisioned with two environments: 'test' and 'prod'. When deploying an API proxy, you can choose to deploy it to one or both environments.

`{base-path}` and `{resource-path}` are defined when you create the API proxy.

When a request comes in to SAP API Management, SAP API Management parses the URL to direct the request to the correct ProxyEndpoint. For example, the following URL is used to access an API proxy on SAP API Management:

```
http://myOrg-prod.<host:port>/v1/weather/forecastrss
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

If you examine the ProxyEndpoint definition for the API proxy in the figure above, you can see how this URL is parsed by SAP API Management:

The domain portion of the URL, http://myOrg-prod.<host:port>, corresponds to a virtual host on SAP API Management. In the ProxyEndpoint definition above, the API proxy uses the <VirtualHost> tag to reference a virtual host named default. You can have multiple virtual hosts defined in your environment.

A virtual host defines the domains and ports on which an API proxy is exposed. A virtual host also defines whether the API proxy is accessed by using the HTTP protocol, or by the encrypted HTTPS protocol.

The second part of the URL, /v1/weather, is determined by the <BasePath> element in the ProxyEndpoint. The base path must be unique to the API proxy for the environment so that two API proxies do not have the same base path.

The third part of the URL, /forecastrss, is a resource defined by the API proxy with the corresponding Conditional Flow defined by the <Flows>tag.

Determining the URL of the target endpoint

The <RouteRule> tag in a ProxyEndpoint definition determines the target of the API proxy, and is evaluated after all policies in the PreFlow, Conditional Flows, and PostFlow of the ProxyEndpoint are processed.

A ProxyEndpoint can define the target as:

- A direct URL to a backend service.
- A single TargetEndpoint definition.
- Multiple TargetEndpoints where the API proxy delegates the request to a target endpoint based on a condition.
- Null route or target, meaning the request is not forwarded to a target. Instead, all of the processing of the request, and the generation of the response, occurs on SAP API Management.

Direct URL

A ProxyEndpoint can directly invoke a backend service, bypassing any named TargetEndpoint configuration. For example, the following<RouteRule> always makes an HTTP call to http://api.mycompany.com/myAPI:

```
<RouteRule name="default">
  <URL>http://api.mycompany.com/myAPI</URL>
</RouteRule>
```

However, because there is no TargetEndpoint, you can only add policies to the flows defined by the ProxyEndpoint.

Single target

In a single target definition, the ProxyEndpoint references a single TargetEndpoint definition by name, as shown in the figure above:

```
<RouteRule name="default">
  <TargetEndpoint>default</TargetEndpoint>
</RouteRule>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

All requests to this API proxy are directed to the same TargetEndpoint definition. The <URL> tag in the TargetEndpoint determines the location of the backend service. in the figure above, the target URL is http://weather.yahooapis.com.

Conditional targets

The <RouteRule> tag lets you direct a request to a target based on a condition. You can use flow variables, query parameters, HTTP headers, message content, or contextual information such time of day and locale to determine the target endpoint. For example, you might include a geographical area, such as US and UK, in a request URL. You can then route a request to a target endpoint based on the region.

The following route rule evaluates an HTTP header in a request. If the HTTP header routeTo has the value TargetEndpoint1, then the request is forwarded to the TargetEndpoint named TargetEndpoint1. If not, then the request is forwarded to TargetEndpoint2.

```
<RouteRule name="MyRoute">
  <Condition>request.header.routeTo = "TargetEndpoint1"</Condition>
  <TargetEndpoint>TargetEndpoint1</TargetEndpoint>
</RouteRule>
<RouteRule name="default">
 <TargetEndpoint>TargetEndpoint2</TargetEndpoint>
</RouteRule>
```

If you have multiple route rules, create one as the 'default', that is, as a route rule with no condition. Ensure that the default route rule is defined last in the list of conditional Routes because rules are evaluated top-down in the ProxyEndpoint.

Null route

A null route supports scenarios in which the request message does not need to be forwarded to a TargetEndpoint. This is useful when the ProxyEndpoint performs all of the necessary processing, for example by using JavaScript to call an external service.

The following example defines a null route:

```
<RouteRule name="GoNowhere"/>
```

3.6 Understanding the handling of request and response data

When you make a request to an API proxy, you can pass any or all of the following information, depending on the way the API proxy is configured:

- Request headers
- Query params
- Form data
- XML or JSON payloads
- Resource URIs

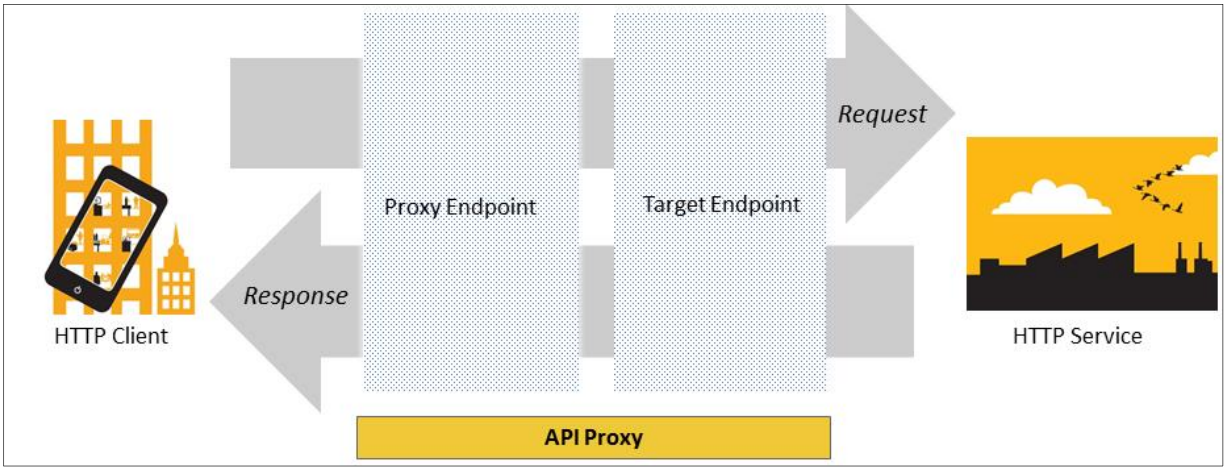
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

By default, all data in a request is passed unchanged from the ProxyEndpoint to the TargetEndpoint. Therefore, when the TargetEndpoint makes the request to the backend server, all information in the original request is passed to the backend service.

The same is true for the response received by SAP API Management from the backend service. By default, all data received in the response is passed unchanged to the app that originated the request.

How request data passed to the backend server?

The following image shows an API proxy definition:



For this API proxy:

- API proxy virtual host: default
- Domain defined by the virtual host: http://myOrg-prod.<host:port>
- Proxy base path: /v1/weather
- TargetEndpoint specifid by route rule: default
- Target URL: http://weather.yahooapis.com

A client app then makes a GET request to the API proxy by using the following cURL command:

```
curl -X GET http://myOrg-prod.<host:port>/v1/weather/forecastrss?w=12797282
```

Notice that this request contains the resource "forecastrss" and one query param, "w". SAP API Management parses the request as shown below and assigns parts of the request to flow variables:

```
{request.verb} {proxy.basepath}/{proxy.pathsuffix}?{request.querystring}
```

The flow variables are set with the following values:

- **request.verb: GET**
- **proxy.basepath: /v1/weather**
- **proxy.pathsuffix: forecastrss**
- **request.querystring: w=12797282**

> **Note**
>
> There are many different flow variables created during the processing of a request and response.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The TargetEndpoint then makes a request to the backend service using information from the request:

```
{request.verb} {target.basepath}/{proxy.pathsuffix}?{request.querystring}
```

Notice how the resource and query params specified in the request are automatically included in the request to the backend server. From the definition of the TargetEndpoint, the request then has the form:

```
curl -X GET http://weather.yahooapis.com/forecastrss?w=12797282
```

Like query params, any headers or form params that you include in the request to the API proxy are passed on to the backend server. For example, you make the request below that includes a header:

```
curl -X GET -H 'Content-type:application/xml' http://myOrgprod-
<host:port>/v1/weather/forecastrss?w=12797282
```

Or a request in the form below to include a header and form data:

```
curl -X POST -H "Content-type:application/json" -d \
'{"email" : "janetutorialxml@example.com",
  "firstName" : "Jane",
  "lastName" : "Tutorial",
  "userName" : "jtutorialxml"
}' \
http://myOrg-prod.<host:port>/v1/register/user
```

In both examples, the headers and form data are passed unchanged to the backend service. The headers are represented by flow variables such asrequest.headers.count and request.headers.names. The form data is represented by flow variables such as request.formparam.countand request.formparam.names.

How is response data returned?

By default, all data received by SAP API Management from the backend service in the response is passed unchanged to the app that originated the request. As described above for the request, the data returned in the response is accessible through flow variables on SAP API Management.

Manipulating request and response data in an API proxy

There are many times where you want to modify request data before sending it to he backend server. For example:

- To remove security information used by SAP API Management to validate requests. That information is not required by the backend service.
- To add data sent to the backend service, for example to track users or to gather analytics.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- To conditionally process the request based on request data. For example, an API proxy can have multiple TargetEndpoints. The TargetEndpoint used by the request is determined by request data. You then strip that data from the request before sending it to the backend service.

The same is true for data in the response. As part of processing the response, the API proxy might want to modify the data before returning it to the requesting app.

SAP API Management defines several policies that you can use to process the request and response data. These policies include:

- Assign Message policy: Creates or modifies HTTP request or response messages during an API proxy flow. Also creates and populates new flow variables.
- Extract Variables policy: Extract content from messages, including headers, URI paths, payloads, and query parameters, for use in a condition statement. The policy then applies a text pattern to message content and upon finding a match sets a designated variable.
- JSON to XML policy and XML to JSON policy: Converts messages from JavaScript Object Notation (JSON) to the extensible markup language (XML) format, or vice versa.
- Java Callout policy, JavaScript policy, Python Script policy, Regular Expression Protection policy: These policies let you write a script to access flow variables containing request and response data.

3.7 API development lifecycle

Every organization has a unique software development lifecycle (SDLC). It is often necessary to synchronize and align API proxy deployment with the same processes that you use today for developing, testing, and deploying other applications.

API Services provides tools and RESTful APIs that enable you to integrate API proxy deployment and management into your organization's SDLC. A common usage of the RESTful API is to write scripts or code that programmatically deploy API proxies, or that migrate API proxies from one environment to another, as part of a larger automated process that also deploys or migrates other applications. API Services makes no assumptions about your SDLC (or anyone else's, for that matter). Rather, it exposes atomic functions that can be coordinated by your development team to automate and optimize your API development lifecycle.

API Services APIs are documented in the API Reference.

3.7.1 Environments

Every organization on SAP API Management has at least two deployment environments that are available for API proxies: 'test' and 'prod'. The distinction between the two environments is arbitrary — each environment is simply identified by a different set of network addresses (URLs). The goal is to provide you with a domain in which you can build and verify API proxies before the API is exposed to external developers.

You can leverage these environments to synchronize API proxy development processed with your SDLC. Each environment is defined by a network address, enabling you to segregate traffic between the API proxies that you are working on, and those that are being accessed by apps at runtime. The network addresses available for each environment is defined in the set of VirtualHosts available in that environment.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Inbound, server SSL is automatically enabled for each environment. Two VirtualHosts are pre-defined in each environment: default and secure. Default defines an HTTP address, while secure defines an HTTP/S address, with pre-configured server-side SSL. In an API proxy configuration, you indicate which VirtualHosts the ProxyEndpoint should listen on. When promoting to prod, you typically disable HTTP by removing the defaultVirtualHost from the API proxy configuration.

For example, the following ProxyEndpoint listens on HTTP and HTTPS.

```
<HTTPProxyConnection>
  <BasePath>/v0/weather</BasePath>
  <Properties/>
  <VirtualHost>default</VirtualHost>
  <VirtualHost>secure</VirtualHost>
</HTTPProxyConnection>
```

By deleting the default VirtualHost from the ProxyEndpoint configuration, you create an API proxy that listens only on HTTPS and not on HTTP.

```
<HTTPProxyConnection>
  <BasePath>/v0/weather</BasePath>
  <Properties/>
  <VirtualHost>secure</VirtualHost>
</HTTPProxyConnection>
```

You can see which VirtualHosts are available in an environment by selecting *Environments* in the management UI main menu.

Environments also provide segregation of data and resources. You can, for example, set up different caches in test and prod, which can be accessed only by API proxies executing in that environment. Additionally, API keys that are issued in the test environment are not valid in the prod environment, and vice-versa.

3.7.2 Deploying API proxies to environments

When you create an API proxy you'll need to decide which environment you'll be working in. You can choose to create a new API proxy on production, but that is not recommended as you may expose an API to developers before it's ready. In general you want to start by creating an API proxy in testwhich, after testing, you then 'promote' to prod.

> **Note**
>
> Depending on your role, you may not be able to deploy to all environments. For example, the 'user' role can only deploy to the testenvironment. If you're an administrator you can deploy to any environment.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.7.3 Iterative development in test

As you work on an API proxy, API Services saves iterations of your configuration as revisions. When you deploy an API proxy, you choose a specific revision to deploy. Typically, you deploy the most recent revision, and, if necessary, revert to the previous revision number. You can choose where to deploy those revisions. For instance, you can promote a revision to prod to allow developers to start working with your API. At the same time, you may be iterating multiple revisions on test, where you're adding features or fine-tuning policies. Then, when you're ready, you can deploy the new revision to prod, overwriting the existing revision on that environment. Using this method, you can always have a live revision of your API available to developers while you're developing.

3.7.4 Promotion to prod

When an API proxy has been fully implemented and tested, it is ready to be promoted to 'prod'. The revision of the API proxy in test will be used to overwrite the revision of the API proxy deployed on prod.

API Services provides capabilities to ensure seamless deployment of API proxies, minimizing the impact on apps and end users during the deployment procedure.

Scripting deployment

The SAP API Management UI enables you to deploy API proxies to prod directly from the API proxy builder. However, in many situations the requirements for security, reliability, and consistency will mandate that development teams script deployment procedures. To do so, you can write code and scripts that invoke the RESTful API exposed by API Services.

Environment resources

For additional control during promotion, it is recommended that you only iterate on API proxies in test, and make as few changes as necessary to API proxies deployed in prod.

To do so, you need to ensure that certain resources associated with each environment are configured in such a way that they can remain static in an API proxy configuration.

- Target URLs: It is common for API proxies to call different backend URLs during testing and production. You can use TargetServer configurations to create environment-independent TargetEndpoint configurations.
- Caches and Key/value maps: Both persistence resources are scoped by environment. You should ensure that naming conventions are used to enable API proxies to store data without requiring configuration changes during promotion.
- ServiceCallout targets: Service callouts may use different targets depending on the environment, if, for example, a ServiceCallout in the test environment consumes a demo service.

To make API proxy configurations environment-independent, you can also use conditional statements. Conditional statement built with theenvironment.name variable can be used to evaluate the current environment before enforcing a policy or before routing to a URL on the backend.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.8 Basic terminology

There are several concepts that are common ideas with a unique meaning in SAP API Management.

Term	Definition
API	An 'application programming interface' is an architecture that makes it easy for one application to 'consume' capabilities or data from another application. By defining stable, simplified entry points to application logic and data, APIs enable developers to easily access and reuse application logic built by other developers. In the case of 'Web APIs', that logic and data is exposed over the network. Since applications that consume APIs are sensitive to changes, APIs also imply a 'contract'. The contract provides some level of assurance that, over time, the API will change in a predictable manner.
API proxy	A facade on SAP API Management for one or more APIs, generic HTTP services, or applications (such as Node.js). An API proxy is implemented as a set of configuration files, policies, and code that rely on a set of resources provided by SAP API Management. API proxies can be generated and configured using the SAP API Management UI, or they can be implemented locally in a text editor or IDE. The facade provided by an API proxy decouples the developer-facing API from 'backend' services, shielding developers from code changes and enabling innovation at the SAP API Management without impacting your internal development teams. As development teams make backend changes, developers continue to call the same interface uninterrupted. SAP API Management enables you to expose multiple interfaces to the same API, freeing you to customize the signature of an API to meet the needs of various developer niches simultaneously.
API base path and resources	APIs defined by network addresses and URIs. An API is made up of a 'base path' and a set of 'API resources'. Every API proxy defines a base path and, optionally, multiple API resource paths. You can think of an API simply as a set of URIs, all of which share a common base path. To make it easier to manage your APIs, SAP API Management augments these raw URIs with display names and descriptions. SAP API Management enables you to attach policies and code to URIs, enabling fine-grained control and management of the behavior of your APIs.
API product	A collection of API resources (URIs) combined with a quota, or 'service plan', which is published to app developers at design time. API products can in turn be bundled into API packages for monetization. An API key is bound to one or more API products, enforcing a binding between an app and the bundle of URIs that the app is permitted to consume.
API package	A collection of API products that are presented to developers as a bundle, and typically associated with a rate plan defined in monetization.
app	An abbreviation of 'application'. The term 'app' has come to refer to mobile applications that consume APIs. Developers implement apps in a variety of programming languages, and using various technologies and platforms. Developers who want to consume APIs register apps in an API provider's organization on SAP API Management. When the app is registered, SAP API Management generates an API key and secret that identify the app. The developer embeds the API key in the app, which presents the key when

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Term	Definition
	making requests. API Services implements security around the API key through direct API key validation or through OAuth.
environment	A runtime execution context for API proxies. An API proxy must be deployed to an environment before the API it exposes is accessible over the network. By default, organizations are provisioned with two environments: 'test' and 'prod'. The 'test' environment is typically used for deploying API proxies during development. The 'prod' environment is typically used for promoting API proxies from the test environment after they have been fully developed and tested.
organization	A container for all the objects in an SAP API Management account, including API proxies, API products, API packages, apps, and developers. A user account is required for each organization for which you are a member. (Most users will have an account in only one organization.) You need to supply your credentials (username and password) and the name of your organization with each API request you submit.
policy	A processing step that executes as an atomic, reusable unit of logic within an API proxy processing flow. Typical policy-based functionality includes transforming message formats, enforcing access control, calling remote services for additional information, masking sensitive data from external users, examining message content for potential threats, caching common responses to improve performance, and so on. Policies may be conditionally executed based on the content or context of a request or response message. For example, a transformation policy may be executed to customize a response format if the request message was sent from a smartphone.
API resource path	A RESTful concept, a resource path is a uniform resource identifier (URI) that identifies the network path to a given resource.
version	The version of the developer-facing API interface. For example, pivotaltracker.com/services/v3, or <host:port>/v1. This term is distinguished from 'revision', which is the numbered, version-controlled package of configuration and policies bundled into an API Proxy. API interfaces have versions; API proxies have revisions.
revision	A numbered, version-controlled package of configuration and policies bundled into an API Proxy. This term is distinguished from 'version', which is the developer-facing API interface. See version above.

3.9 Best practices for API proxy design and development

The purpose of this document is to provide a set of standards and best practices for developing with SAP API Management. The topics that are covered here include design, coding, policy use, monitoring, and debugging. The information has been gathered by the experience of developers working with SAP API Management to implement successful API programs. This is a living document and will be updated from time to time.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.9.1 Development standards

Comments and Documentation

- Provide inline comments in the ProxyEndpoint and TargetEndpoint configurations. Comments enhance readability for a Flow, especially where policy file names are not sufficiently descriptive to express the underlying functionality of the Flow.
- Make comments useful. Avoid obvious comments.
- Use consistent indentation, spacing, vertical alignment, etc.

Framework-style coding

Framework-style coding involves storing API proxy resources in your own version control system for reuse across local development environments. For example, to reuse a policy, story it in source control so that developers can sync to it and use it in their own proxy development environments.

- To enable DRY ("don't repeat yourself"), where possible, policy configurations and scripts should implement specialized, reusable functions. For example, a dedicated policy to extract query parameters from request messages could be called `ExtractVariables.ExtractRequestParameters`. A dedicated policy to inject CORS headers could be called `AssignMessage.SetCORSHeaders`. Those policies could then be stored in your source control system and added to each API proxy that needs to extract parameters or set CORS headers, without requiring you to create redundant (and hence less manageable) configurations.
- Clean up unused policies and resources (JavaScript, Java, XSLT, etc.) from API proxies, especially large resources that have the potential to slow down import and deploy procedures.

Naming Conventions

- The Policy `name` attribute and the XML policy file name must be identical.
- The Script and ServiceCallout policy name attribute and the name of the resource file should be identical.
- DisplayName should accurately describe the policy's function to someone who has never worked with that API proxy before..
- Name policies according to their function, for example AssignTargetAuthHeader or AssignMessage.TargetAuthHeader,RaiseFault.InvalidUser.
- Use proper extensions for resource files, .js for JavaScript, .py for python, and .jar for Java JAR files.
- Variable names should be consistent. If you choose a style, such as camelCase or under_score, use it throughout the API proxy.
- Use variable prefixes, where possible, to organize variables based on their purpose, for example, Consumer.username and Consumer.password.

3.9.2 API proxy development

Initial Design Considerations

Leverage SAP API Management policies and functionality wherever possible to build API proxies. Avoid coding all proxy logic in JavaScript, Java, or Python resources.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Do not call the SAP API Management API from inside API proxies. The SAP API Management API is used for administrative management purposes, not API flow logic. Policies are provided for interaction with API Services entities, such as developers, apps, access tokens and so on.

Construct Flows in an organized manner. Multiple Flows, each with a single condition, are preferable to multiple conditional attachments to the same PreFlow and Postflow.

As a 'failsafe', create a default API proxy with a ProxyEndpoint BasePath of `/`. This can be used to redirect base API requests to a developer site, to return a custom response, or perform another action more useful than returning the default CLASSIFICATION_ERROR.

Use TargetServer resources to decouple TargetEndpoint configurations from concrete URLs, supporting promotion across environments.

If you have multiple RouteRules, create one as the 'default', that is, as a RouteRule with no condition. Ensure that the default RouteRule is defined last in the list of conditional Routes. RouteRules are evaluated top-down in ProxyEndpoint.

3.9.3 Message payload size

To prevent memory issues in SAP API Management, message payload size is restricted to 3MB.

Exceeding those sizes results in a protocol.http.TooBigBody error.

Following are the recommended strategies for handling large message sizes in SAP API Management:

- Stream requests and responses. Note that when you stream, policies no longer have access to the message content.
- In SAP API Mangement deployments, edit the message processor `http.properties` file to increase the limit in the `HTTPResponse.body.buffer.limit` parameter. Be sure to test before deploying the change to production.

3.9.4 Fault Handling

- Leverage FaultRules to handle all fault handling. (RaiseFault policies are used to stop message Flow and send processing to the FaultRules Flow.)
- Within the FaultRules Flow, use AssignMessage policies to build the fault response, not RaiseFault policies. Conditionally execute AssignMessage policies based on the fault type that occurs.
- Always includes a default 'catch-all' fault handler so that system-generated faults can be mapped to customer-defined fault response formats.
- If possible, always make fault responses match any standard formats available in your company or project.
- Use meaningful, human-readable error messages that suggest a solution to the error condition.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.9.5 Persistence

Key/Value Maps

- Use key/value maps only for limited data sets. They are not designed to be a long-term data store.
- Consider performance when using key/value maps as this information is stored in the Cassandra database.

Response Caching

- Do not populate the response cache if the response is not successful or if the request is not a GET. Creates, updates, and deletes should not be cached. <SkipCachePopulation>response.status.code != 200 or request.verb != “GET”</SkipCachePopulation>
- Populate cache with a single consistent content type (for example, XML or JSON). After retrieving a responseCache entry, then convert to the needed content type with JSONtoXML or XMLToJSON. This will prevent storing double, triple, or more data.
- Ensure that the cache key is sufficient to the caching requirement. In many cases, the request.querystring can be used as the unique identifier.
- Do not include the API key (client_id) in the cache key, unless explicitly required. Most often, APIs secured only by a key will return the same data to all clients for a given request. It is inefficient to store the same value for a number of entries based on the API key.
- Set appropriate cache expiration intervals to avoid dirty reads.
- Ideally, the response cache policy should be attached to the ProxyEndpoint response PostFlow, before any format translations (XMLToJSON, JSONToXML)
- The response cache policy to lookup the cache entry should occur in the ProxyEndpoint request PreFlow. Avoid implementing too much logic, other than cache key generation, before returning a cache entry. Otherwise, the benefits of caching are minimized.
- In general, you should always keep the response cache lookup as close to the client request as possible. Conversely, you should keep the response cache population as close to the client response as possible.

3.9.6 Policy and custom code

Policy or custom code?

- Use built-in policies first and foremost (when possible). SAP API Management policies are hardened, optimized, and supported. For example, use the standard AssignMessage and ExtractVariables policies instead of JavaScript (when possible) to create payloads, extract information from payloads (XPath, JSONPath), etc.
- JavaScript is preferred over Python and Java. However, if performance is the primary requirement, Java should be used over JavaScript.

JavaScript

- Use JavaScript if it’s more intuitive than SAP API Management policies (for example, when setting target.url for many different URI combinations).
- Complex payload parsing such as iterating through a JSON object and Base64 encoding/decoding.
- JavaScript policy has a time limit, so infinite loops are blocked.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- Always use JavaScript Steps and put files in jsc resources folder. JavaScript Policy type pre-complies the code at deployment time.

Java

- Use Java if performance is the highest priority, or if the logic cannot be implemented in JavaScript.
- Include Java source files in source code tracking.

Python

- Do not use Python unless absolutely required. Python scripts can introduce performance bottlenecks for simple executions, as it is interpreted at runtime.

3.9.7 Script Callouts (Java, JavaScript, Python)

- Use a global try/catch, or equivalent.
- Throw meaningful exceptions and catch these properly for use in fault responses.
- Throw and catch exceptions early. Do not use the global try/catch to handle all exceptions.
- Perform null and undefined checks, when necessary. An example of when to do this is when retrieving optional flow variables.
- Avoid making HTTP/S requests inside of a script callout. Instead, use the SAP API Management ServiceCallout policy as the policy handles connections gracefully.

JavaScript

- JavaScript on the API Platform supports XML via E4X.

Java

- When accessing message payloads, try to use context.getMessage() vs. context.getResponseMessage or context.getRequestMessage. This ensures that the code can retrieve the payload, in both request and response flows.
- Import libraries to the SAP API Management organization or environment and do not include these in the JAR file. This reduces the bundle size and will let other JAR files access the same library repository.
- Import JAR files using the SAP API Management resources API rather than including them inside the API proxy resources folder. This will reduce deployment times and allow the same JAR files to be referenced by multiple API proxies. Another benefit is class loader isolation.
- Do not use Java for resource handling (for example, creating and managing thread pools).
- Use @IOIntensive annotation for all Java classes. This will cause the class to run in a separate thread pool. Otherwise it runs in an I/O thread, which is in the core thread pool and has the potential to block the CPU.

Python

- Throw meaningful exceptions and catch these properly for use in SAP API Management fault responses

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.9.8 Accessing entities

AccessEntity Policy

For better performance, look up apps by uuid instead of app name.

Logging

Use a common syslog policy across bundles and within the same bundle. This will keep a consistent logging format.

3.9.9 Monitoring

Cloud customers are not required to check individual components of SAP API Management (Routers, Message Processors, etc.). SAP API Management's Global Operations team is thoroughly monitoring all of the components, along with API health checks, given health check requests by the customer.

SAP API Management Analytics

Analytics can provide non-critical API monitoring as error percentages are measured.

Trace

The trace tool in the SAP API Management UI is useful for debugging runtime API issues, during development or production operation of an API.

3.9.10 Security

- Use IP address restriction policies to limit access to your test environment. Whitelist the IP addresses of you development machines or environments.
- Always apply content protection policies (JSON and or XML) to API proxies that are deployed to production.

3.9.11 ServiceCallouts

- Don't make ServiceCallouts to other API proxies in the same organization, including recursive callouts back into the same API proxy.
- Build a ServiceCallout request message using the AssignMessage policy, and populate the request object in a message variable. (This includes setting the request payload, path, and method.)
- The URL that is configured within the policy requires the protocol specification, meaning the protocol portion of the URL, https:// for example, cannot be specified by a variable. Also, you must use separate variables for the domain portion of the URL and for the rest of the URL. For example: https://{domain}/{path}
- Store the response object for a ServiceCallout in a separate message variable. You can then parse the message variable and keeps the original message payload intact for use by other policies.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10 Implementing API Proxies

3.10.1 Building a Simple API Proxy

SAP API Management enables you to quickly expose backend services as APIs. You do this by creating an API proxy that provides a facade for the backend service that you want to expose. You only need to provide the network address for the backend service, along with some information that API Platform uses to model the API that will be exposed to developers.

The API proxy that you create on SAP API Platform decouples your backend service implementation from the API that developers consume. This shields developers from future changes to your backend services. As you update backend services, developers, insulated from those changes, can continue to call the API uninterrupted.

This topic shows you how to create a simple API proxy that runs on SAP API Management.

3.10.1.1 Creating an API Proxy in the SAP API Management UI

The easiest way to create an API proxy is using the API proxy builder tool, which is available in the SAP API Management UI.

1. Log in into the SAP API Management UI.
2. From the main menu, select APIs > API Proxies.

   A list of all API proxies in your organization displays.
3. To create a new API proxy, select add (+) API Proxy button.

   The New API Proxy form is displayed.

   Use this form to provide API Platform with the information it needs to generate an API proxy and, optionally, to add some minimal features to the API that is exposed.

   This form enables you to create an API proxy from the following sources:

   - An existing HTTP service (this could be a JSON or XML API)
   - An existing API proxy bundle
   - A WSDL file
   - An API proxy with no API backend ("No target")
   - A Node.js application

   The following sections describe how to create an API proxy using each source.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.1.2 Creating an API Proxy for an HTTP Service

API Platform generates API proxies based on two pieces of information:

- The URL of the backend service
- The URI path that uniquely identifies the API that will be exposed by the API proxy to consumer apps

The backend service URL typically represents a service-enabled application owned by your organization. It can also point to a publicly available API. The API or service can be under your control "(for example, any SAP Gateway Odata service, Enterprise SOAP Service) or it can be a third-party API or service.

To add an API proxy for an existing HTTP service:

1. Under Choose Your Starting Point:
   - Select Existing Backend service.
   - Enter the Backend Service URL, for example, `http://weather.yahooapis.com`. The Backend Service URL defines the target URL that SAP API Management invokes on behalf of apps. You can add any URL that is accessible to the SAP API Management server.
2. Under Identify Your API Proxy:
   - Enter a descriptive name for your API, such as weather.
   - Enter a Project Base Path.

The Project Base Path is a URI fragment that uniquely identifies the API that is exposed by this API proxy. API Platform uses the Base Path URI to match and route incoming request messages to the proper API proxy. (The Base Path is appended to the domain of the API) It's a best practice to include a version number in the project name, for example, `/v1/weather`. This will determine how your API is invoked by consumer apps.

   - Optionally, add a brief description of the API.
3. Optionally, add security and CORS support to your API.
   See Adding CORS Support to an API proxy for more information.
4. Select Build
   In response, you should see an acknowledgment that your new API proxy was successfully generated. API Services automatically deploys the newly generated API proxy to the 'test' environment in your organization.
5. The new API is immediately available to be invoked.

3.10.1.3 Importing API Proxies from an API Proxy Bundle

You can use this capability to import any API proxies that you or other developers have configured.

1. In the New API Proxy dialog box, under Choose Your Starting Point section:
   1. Select the API Bundle.
   2. Select Choose File.
   3. Navigate to the ZIP file containing the API proxy configuration
2. Under Identify Your API Proxy, enter a descriptive name for your API.
3. Select Build
   In response, you should see an acknowledgment that your new API proxy was successfully imported. API Platform automatically deploys the imported API proxy to the 'test' environment in your organization. The API exposed by the API proxy is available to be invoked.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.1.4 Copying and backing up and API proxy

You can copy an existing API proxy to a new API proxy, or backup and existing API proxy to a set of XML files as an API proxy bundle. Once exported to a bundle, you can import the API proxy to a new proxy, as described above.

To copy an existing API proxy to a new API proxy:

1. In the main menu of the management UI, choose APIs to display the API Proxies page.
2. Choose the name of the proxy in the API Proxies table.
3. In the upper-left side of the API proxy detail page, choose Project > Save as New API proxy.
4. Specify the name of the new API proxy.
5. Select Add.

To backup and existing API proxy to an API proxy bundle:

1. In the main menu of the management UI, choose APIs to display the API Proxies page.
2. Choose the name of the proxy in the API Proxies table.
3. In the upper-left side of the API proxy detail page, choose Project > Download Current Revision to download a ZIP file of the bundle.

3.10.1.5 Consuming one proxy from another proxy

You can specify that one proxy is the target endpoint of another, effectively connecting the two proxies in a proxy chain.

You might find this useful when you have a proxy that offers some discrete low-level functionality that one or more other proxies will consume. For example, a proxy that exposes create/read/update/delete operations with a backend data store could be the target proxy for multiple other proxies that expose the data to clients.

To minimize network overhead when calling one proxy from another, you can specify that one proxy is the local target endpoint of the other. This kind of connection, called proxy chaining, gives you a local connection between two proxies. This local connection is more efficient because it bypasses network features such as load balancers, routers, and message processors. When you specify a proxy as a local target endpoint, processing flows to that proxy as it would to another kind of target.

Note:

Proxies you want to connect in this way must be in the same organization and environment. When the proxy you're connecting with is outside this scope, consider connecting to it as an external resource using the `HTTPTargetConnection` element.

You connect proxies by specifying that one is a local target endpoint of the other. You can create a local connection between proxies in two ways:

- By specifying the name of the target proxy and a ProxyEndpoint name
- By specifying a path to the target proxy endpoint

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

You connect target proxies within a `TargetEndpoint` configuration, using a `LocalTargetConnection` element, as described below.

Caution:

When chaining API proxies, take care to avoid "infinite loop" recursive callouts back into the same API proxy, which could result in system overload.

3.10.1.5.1 Connecting proxies by proxy name

You can specify the target proxy by name. You might find that this is most useful when you're creating the connection from the start and developing the proxies together. If you don't know the name (or the name might change), consider connecting with the target proxy's endpoint path, as described below.

When you connect to a target proxy by name, you specify the proxy's name and the name of its ProxyEndpoint.

The following example specifies a target proxy called `data-manager`, along with the `ProxyEndpoint` name exposed by `data-manager`.

```
<TargetEndpoint name="datamanager">
    <PreFlow name="PreFlow">
        <!-- PreFlow policies -->
    </PreFlow>
    <PostFlow name="PostFlow">
        <!-- PostFlow policies -->
    </PostFlow>
    <LocalTargetConnection>
        <APIProxy>data-manager</APIProxy>
        <ProxyEndpoint>default</ProxyEndpoint>
    </LocalTargetConnection>
</TargetEndpoint>
```

3.10.1.5.2 Connecting proxies by path

You can specify the target proxy by its endpoint path. You might want to do it this way when you don't know the proxy name, or when the name might change.

If your proxy is simply the consumer of the target proxy -- such as when you're not developing both -- the path might be the most reliable way to connect. For example, if the proxy you're connecting to is developed and maintained by another team, you might want to connect using a reliable endpoint path.

The following example specifies a target proxy at `/v1/streetcarts/foodcarts/data-manager`, where the host is assumed to be the same as the current proxy

```
<TargetEndpoint name="datamanager">
    <PreFlow name="PreFlow">
        <!-- PreFlow policies -->
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    </PreFlow>
    <PostFlow name="PostFlow">
        <!-- PostFlow policies -->
    </PostFlow>
    <LocalTargetConnection>
        <Path>/v1/streetcarts/foodcarts/data-manager</Path>
    </LocalTargetConnection>
</TargetEndpoint>
```

3.10.1.5.3 Connecting proxies with the management console

You can create proxy chaining connections using the SAP API Management Edge console.

1. Open the proxy that will be consuming the target proxy.
2. In the Navigator, click the plus sign next to Target Endpoints.
3. In the New Target Endpoint dialog, enter the name of the target endpoint.
4. Beneath the Target Endpoint Name box, select one of the following:

- Proxy Chaining to select from a list of proxies already in the organization and environment.
  1. In the Proxy Name dropdown, select the target proxy.
  2. In the Proxy Endpoint box, enter the target proxy endpoint path you want to connect to.
- Path Chaining to enter the target proxy base path, such a `/mypath/myproxy/myendpoint`.

5. Click Add.

3.10.1.6 Exposing a SOAP-based Web Service as an API Proxy

You can create two kinds of SOAP proxies in SAP API Management. One generates a RESTful interface to the backend SOAP service and the other performs a "pass through" of the SOAP message to the backend. For details, see Exposing a SOAP service as an API proxy.

3.10.1.7 Adding Security and CORS support to your API

When you add an API proxy for an existing backend service or import an existing API bundle, the New API Proxy page displays an Add Features section in which you can add features to your API such as security and support for CORS.

> **Note**
>
> If you import an API bundle, then you are responsible for configuring security in the bundle. The UI does not prompt you to add security when importing the bundle.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

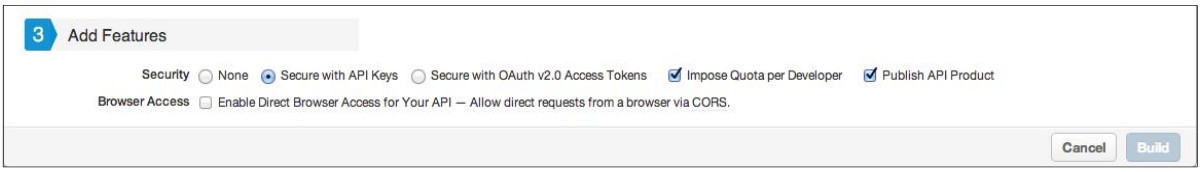
3.10.1.7.1 Adding Security

Check Security with API Keys to add simple API key verification to the API proxy that you are defining. In response, the SAP API Management adds a VerifyAPIKey policy and an AssignMessage policy to your API proxy. The VerifyAPIKey policy validates API keys presented by requesting apps. The AssignMessage policy strips the API key, supplied in the API call as a query parameter, from the request forwarded to the backend server.

If you select Secure with OAuth v2.0 Access Tokens, SAP API Management will automatically add two policies to your API proxy: one policy to verify an access token and another policy to strip the access token from the message before forwarding it to your backend service. To learn how to obtain an access token, see OAuth.

When you check Secure with API Keys, the New API Proxy page displays an additional checkbox:

Impose Quota per Developer. Check this to add a Quota policy to your API proxy that enforces a limit on the traffic to your API from individual apps.



3.10.1.7.2 Adding Support for CORS

CORS (Cross-origin resource sharing) is a standard mechanism that allows a Web browser to make direct requests to another domain. The CORS standard defines a set of HTTP headers that Web browsers and servers use to implement cross-domain communication.

You can add support for CORS to your API by selecting the Enable Direct Browser Access for your API checkbox under Add Features. For more detailed information about CORS support, including adding CORS preflight support to a proxy, see Adding CORS Support to an API proxy.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.2 Flows Configuration

Flows are the basic building blocks of API proxies. Flows enable you to program the behavior of an API by letting you configure the sequence in which policies and code are executed by an API proxy.

For a conceptual overview of flows, see Understanding flows.

3.10.2.1 Attaching Policies to Flows

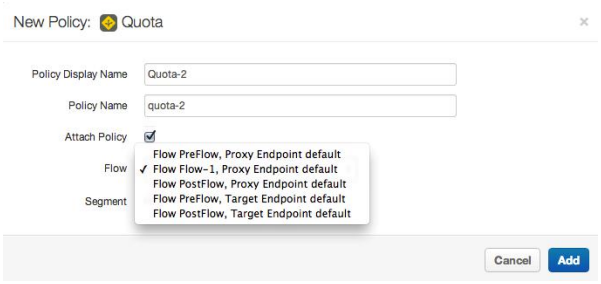
SAP API Management comes with many different types of predefined policies to implement security, manage traffic, and manipulate messages. In addition, policies let you add your own custom code to completely customize message processing.

For example:

- Attach an OAuth security policy to the request PreFlow of the ProxyEndpoint. Because the ProxyEndpoint's request PreFlow is the first flow in the pipeline, you can immediately reject a request if it violates your security policies.
- Attach a JSON to XML conversion policy to the response PostFlow of the TargetEndpoint to convert a response from JSON to XML.
- Attach a JavaScript policy to a Conditional Flow of the ProxyEndpoint to execute JavaScript code to process the request

Once you have created a conditional flow, it is available for Policy attachment. When you select New > Policy after you create the flow, you will see an additional option in the list of flows. As shown below, when adding a Quota policy, you now have the option to attach the Policy to the flow called Flow-1, which you just created.

By attaching the Policy to Flow-1, you are configuring the API proxy to enforce the Quota Policy only for requests submitted using the GET verb. The Quota will not be enforced for POSTs, PUTs, etc.



The following ProxyEndpoint definition shows the underlying source XML with a policy named quota-2 on the request PreFlow of the ProxyEndpoint:

```
<ProxyEndpoint name="default">
  <PreFlow name="PreFlow">
    <Request>
      <Step>
        <Name>quota-2</Name>
      </Step>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    </Request>
    <Response/>
  </PreFlow>
  <Flows/>
  <PostFlow>
    <Request/>
    <Response/>
  </PostFlow>
  ...
</ProxyEndpoint>
```

3.10.2.2 About Conditional Flows

Any policies attached to the PreFlow or PostFlow are always executed. However, the policies in a conditional flow are executed only if the flow's condition evaluates to true.

During the processing of a request and response, only one conditional flow is executed per segment--the first flow whose condition evaluates to true. That means you can have one conditional flow executed as part of each of the:

- ProxyEndpoint's request pipeline
- TargetEndpoint's request pipeline
- ProxyEndpoint's response pipeline
- TargetEndpoint's response pipeline

For example, the following ProxyEndpoint definition shows a conditional flow that is executed by the ProxyEndpoint on any HTTP GET request to the API proxy:

```
<ProxyEndpoint name="default">
  <PreFlow>
    <Request/>
    <Response/>
  </PreFlow>
  <Flows>
    <Flow name="Flow-1">
      <Condition>request.verb="GET"</Condition>
      <Request/>
      <Response/>
    </Flow>
  </Flows>
  <PostFlow>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <Request/>
    <Response/>
  </PostFlow>
  ...
</ProxyEndpoint>
```

Notice that the condition references the `request.verb` flow variable. A flow variable is named references that hold state information associated with an API transaction processed by SAP API Management. SAP API Management defines many state variables that you can reference.

RESTful services are collections of API resources. An API resource is a URI path fragment that identifies some entity that developers can access by calling your API. For example, if your service backend provides weather reports and weather forecasts, your API might define two conditional flows that map to those API resources: `/reports` and `/forecasts`. When an API call includes one of those resources in the URL, the condition evaluates to true and the logic attached to the conditional flow is executed.

App developers then access your resources by making requests to a URL in the form:

```
http://myAPIs.myCo.com/weather/reports
```

or

```
http://myAPIs.myCo.com/weather/forecasts
```

In an API proxy, you can define a conditional flow that corresponds to a specific resource:

```
<ProxyEndpoint name="default">
  <PreFlow>
    <Request/>
    <Response/>
  </PreFlow>
  <Flows>
    <Flow name="Flow-1">
      <Condition>(proxy.pathsuffix MatchesPath &quot;/reports&quot;)</Condition>
      <Request/>
      <Response/>
    </Flow>
    <Flow name="Flow-2">
      <Condition>(proxy.pathsuffix MatchesPath &quot;/forecasts&quot;)</Condition>
      <Request/>
      <Response/>
    </Flow>
  </Flows>
  <PostFlow>
    <Request/>
    <Response/>
  </PostFlow>
  ...
</ProxyEndpoint>
```

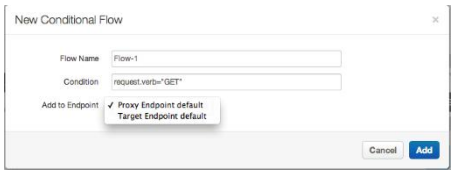

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

In this example, you reference the `proxy.pathsuffix` flow variable, which contains the suffix portion of the URL used to access the API proxy. You can then attach different policies to the conditional flow for each resource.

3.10.2.3 Adding a Conditional Flow

In this brief example, you set up a flow that executes only when the request message is an HTTP GET.

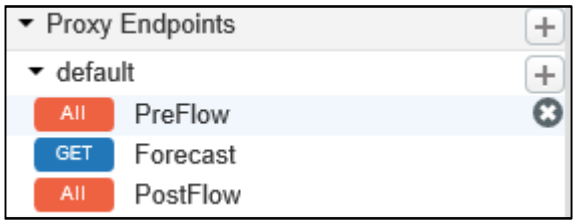
To add a conditional flow, select the Develop view in the API proxy builder. Select New > Conditional Flow.



The New Conditional Flow form enables you name the flow and to add a condition. In this example, you add a condition that evaluates the HTTP of the request message. You add a condition that will evaluate to true if the HTTP verb is GET (as opposed to PUT, POST, etc.)

The form also enables you to add the flow to the ProxyEndpoint named default or the TargetEndpoint named default.

Select the Proxy endpoint default option.



The new flow, called `Flow-1`, now appears in the Navigator menu.

Now observe the XML configuration for the ProxyEndpoint. Select Flow-1 in the Navigator menu.

You will see the following configuration.

```
<PreFlow name="PreFlow">
    <Request/>
    <Response/>
</PreFlow>
<Flows>
    <Flow name="Flow-1">
      <Request/>
      <Response/>
      <Condition>request.verb="GET"</Condition>
    </Flow>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
</Flows>
<PostFlow name="PostFlow">
    <Request/>
    <Response/>
</PostFlow>
```

3.10.3 Flow Variables and Conditions

Conditional statements are a common control structure in all programming languages. Like a programming language, API proxy configuration supports conditional statements for Flows, Policies, and RouteRules. By defining conditional statements, you define dynamic behavior for your API. This dynamic behavior lets you do things like converting XML into JSON only for mobile devices, or routing to a backend URL based on the content type or HTTP verb of the request message.

This topic shows you how to use conditions to dynamically apply API management features at runtime, without writing any code.

3.10.3.1 Configuring Conditional Statements

Conditional behavior is implemented in API proxies by using a combination of conditions and variables. A conditional statement is created using a Condition element. The following is an empty condition:

```
<Condition></Condition>
```

To create a conditional statement, you add a conditional operator and a variable to create the following structure:

```
<Condition>{variable.name}{operator}{"value"}</Condition>
```

Supported conditional operators include = (equals), != (not equal), and > (greater than). For readability, you can also write the conditionals as text: equals, notequals, greaterthan.

When working with URI paths you can use ~/ or MatchesPath. You can also use conditions to evaluate regular expressions.

3.10.3.2 Variables

Conditions do their work by evaluating the values of variables. A variable is a property of an HTTP transaction executed by an API proxy, or a property of an API proxy configuration itself. Whenever an API proxy gets a request from an app, API Platform populates a long list of variables that are associated with things like system time, the app's network information, HTTP headers on messages, the API proxy configuration, policy executions and so on. This creates a rich context that you can use to setup conditional statements.

Variables always use a dotted notation. For example, HTTP headers on the request message are available as variables called request.header.{header_name}. So to evaluate the Content-type header, you could use the variablerequest.header.Content-type. For example request.header.Content-type ="application/json" indicates that the variable value should be JSON.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Imagine that you need to create a conditional statement that will cause a policy to be enforced only when a request message is a GET. To create a condition that evaluates the HTTP verb of a request, you create the conditional statement below. The variable in this condition is request.verb. The value of the variable is GET. The operator is=.

```
<Condition>request.verb = "GET"</Condition>
```

```
<Condition>request.verb equals "GET"</Condition>
```

API Platform uses such a statement to evaluate conditions. The example above evaluates to true if the HTTP verb associated with the request is GET. If the HTTP verb associated with the request is POST, then the statement evaluates to false.

Recommendation

API Platform automatically stores all HTTP headers and query parameters as variables that can be accessed using request.header and request.queryparam variable prefixes.

To enable dynamic behavior, you can attach Conditions to Flows, Steps, and RouteRules.

When you attach a condition to a Flow, you create a 'conditional Flow'. Conditional Flows execute only when the condition evaluates to true. You can attach as many Policies as you want to a conditional Flow. A conditional Flow enables you to craft highly specialized processing rules for request or response messages that meet certain criteria.

For example, to create a Flow that executes only when the request verb is a GET:

```
<Flows>
  <Flow name="ExecuteForGETs">
  <Condition>request.verb="GET"</Condition>
  </Flow>
</Flows>
```

To create one Flow for GETs and another for POSTs:

```
<Flows>
  <Flow name="ExecuteForGETs">
  <Condition>request.verb="GET"</Condition>
  </Flow>
  <Flow name="ExecuteForPOSTs">
  <Condition>request.verb="POST"</Condition>
  </Flow>
</Flows>
```

As shown in the example below, you can apply the condition to the Policy Step itself. The following Condition causes the VerifyApiKey Policy to be enforced only if a request message is a POST.

```
<PreFlow name="PreFlow">
    <Request>
        <Step>
       <Condition>request.verb equals "POST"</Condition>
            <Name>VerifyApiKey</Name>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        </Step>
    </Request>
</PreFlow>
```

Once you have defined such conditional Flows, you can attach Policies to them, enabling an API proxy to enforce one set of policies for GET requests, and another set of policies for POST requests.

Conditionally Converting a Payload from XML to JSON

Let's use a specific example in which you need to transform response message from XML to JSON only for mobile devices. First, create the policy that will convert the XML-formatted response from the Weather API into JSON:

```
<XMLToJSON name="ConvertToJSON">
  <Options>
  </Options>
  <OutputVariable>response</OutputVariable>
  <Source>response</Source>
</XMLToJSON>
```

The policy configuration above tells the API proxy to take the response message, perform a conversion from XML to JSON with default settings, and then write the result to the new response message. (If you are converting a request message from XML to JSON, you simply set both of these values to request.)

Recommendation

The XML To JSON policy type defines a set of reasonable defaults, which means that you only need to add configuration elements to craft XML into specific JSON structures.

Since you want to convert responses from XML to JSON, you need to configure a conditional response Flow to perform the conversion. For example, to convert all responses from XML to JSON before they are returned to the client app, configure the following ProxyEndpoint response Flow.

```
<Flows>
  <Flow name="Convert-for-devices">
    <Response>
      <Step><Name>ConvertToJSON</Name></Step>
    </Response>
  </Flow>
</Flows>
```

When you invoke the API using the standard request, the response is formatted in JSON.

However, your goal is to only convert Weather reports into JSON when the requesting client is a mobile device. To enable such dynamic behavior, you must add a conditional statement to the Flow.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.3.3 Configuring Conditional Behavior

The following example shows a single conditional flow named Convert-for-devices, configured in the ProxyEndpoint response Flow. Add the Condition as an element to the entity to which the condition applies. In this example, the condition is a component of the Flow. Therefore, the Flow will execute whenever the statement evaluates to true.

```
<Flows>
  <Flow name="Convert-for-devices">
  <Condition>(request.header.User-Agent = "Mozilla")</Condition>
    <Response>
      <Step><Name>ConvertToJSON</Name></Step>
    </Response>
  </Flow>
</Flows>
```

For each request received from an app, API Platform stores the values of all HTTP headers present as variables. If the request contains an HTTP header called User-Agent, that header and its value are stored as a variable called `request.header.User-Agent.`

Given the ProxyEndpoint configuration above, API Platform checks the value of the `request.header.User-Agent` variable to see whether the condition evaluates to true.

If the condition does evaluate to true, that is, the value of the variable request.header.User-Agent equalsMozilla, then the conditional Flow executes and the XMLtoJSON policy called ConvertToJSON is enforced. If not, the Flow is not executed, and the XML response is returned unmodified (in XML format) to the requesting app.

Testing the Conditional Flow

In this sample request, the HTTP User-Agent header is set to Mozilla, causing the conditional statement to evaluate to true and the conditional flow Convert-for-devices to execute.

```
$ curl -H "User-Agent:Mozilla" http://<host:port>/weather/forecastrss?w=12797282
```

or, to pretty print where Python is available:

```
$ curl -H "User-Agent:Mozilla" http://<host:port>/weather/forecastrss?w=12797282 |
python -mjson.tool
```

Sample Response:

```
. . .
"yweather_forecast": [
        {
            "code": "11",
            "date": "12 Dec 2012",
            "day": "Wed",
            "high": "55",
            "low": "36",
            "text": "Showers"
         },
         {
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
            "code": "32",
            "date": "13 Dec 2012",
            "day": "Thu",
            "high": "56",
            "low": "38",
            "text": "Sunny"
        }
    ]
  }
. . .
```

A request submitted without the User-Agent header, or with a different value than Mozilla, will result in an XML-formatted response.

```
$ curl http://{org_name}-test.<host:port>/weather/forecastrss?w=12797282
```

The unmodified XML response is returned.

Sample Response:

```
<yweather:forecast day="Wed" date="12 Dec 2012" low="36" high="55" text="Showers"
code="11" /> <yweather:forecast day="Thu" date="13 Dec 2012" low="38" high="56"
text="Sunny" code="32" />
```

3.10.3.4 Sample Conditions

Condition Attached to RouteRule

```
<RouteRule name="default">
 <!--this routing executes if the header indicates that this is an XML call. If
true, the call is routed to the endpoint XMLTargetEndpoint-->
  <Condition>request.header.content-type = "text/xml"</Condition>
  <TargetEndpoint>XmlTargetEndpoint</TargetEndpoint>
</RouteRule>
```

Condition Attached to a Policy

```
<Step>
<!--the policy MaintenancePolicy only executes if the response status code is
exactly 503-->
  <Condition>response.status.code is 503</Condition>
  <Name>MaintenancePolicy</Name>
</Step>
```

Conditional Flow

```
<!-- this entire flow is executed only if the response verb is a GET-->
<Flow name="GetRequests">
  <Condition>response.verb="GET"</Condition>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  <Request>
    <Step>
<!-- this policy only executes if request path includes a term like statues-->
<Condition>request.path ~ "/statuses/**"</Condition>
      <Name>StatusesRequestPolicy</Name>
    </Step>
  </Request>
  <Response>
    <Step>
<!-- this condition has multiple expressions. The policy executes if the response
code status is exactly 503 or 400-->
<Condition>(response.status.code = 503) or (response.status.code = 400)</Condition>
      <Name>MaintenancePolicy</Name>
    </Step>
  </Response>
</Flow>
```

Sample Operators in Conditions

Here are some examples of operators used to create conditions:

- `request.header.content-type = "text/xml"`
- `request.header.content-length < 4096 && request.verb = "PUT"`
- `response.status.code = 404 || response.status.code = 500`
- `request.uri MatchesPath "/*/statuses/**"`
- `request.queryparam.q0 NotEquals 10`

To maintain performance and availability across a diverse base of client apps, it's critical to maintain app traffic within the limits of the capacity of your APIs and backend services. It's also important to ensure that apps don't consume more resources than permitted.

SAP API Management provides three mechanisms that enable you to optimize traffic management to minimize latency for apps while maintaining the health of backend services. Each policy type addresses a distinct aspect of traffic management. In some cases, you might use all three policy types in a single API proxy.

3.10.4 Mapping Conditional Flows to API Resources

RESTful services are collections of API resources. An API resource is a URI path fragment that identifies some entity that developers can access by calling your API. For example, if your service backend provides weather reports and weather forecasts, your API might define two API resources: `/reports` and `/forecasts`.

Recommendation

The term resource is used in the Web Application Description Language (WADL) specification. This is just for convenience. It doesn't necessarily imply any endorsement of WADL. According to WADL, a RESTful API consists of a base path and any number of resource paths. Resources are sometimes called API

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

methods, but the term method can become overloaded, because a RESTful API also has associated HTTP verbs (`GET, PUT, POST, DELETE`) which are also sometimes called methods.

To illustrate API resources, we can use a (very simplified) example drawn from SAP API Management's own Developer Services API. The following snippet of WADL defines two API resources, /developers and /apps. For each API resource, it defines two methods, create and list.

```
<resource path="/developers">
  <method id="createDeveloper" name="POST">
  </method>
  <method id="listDevelopers" name="GET">
  </method>
</resource>
<resource path="/apps">
  <method id="createApp" name="POST">
  </method>
  <method id="listApps" name="GET">
  </method>
</resource>
```

3.10.4.1 How API proxies map to specific backend resources

With an API proxy URL mapped to the base URL of the backend service (when you create the proxy), you can add conditional flows to specific resources, such as the /reports and /forecasts resources mentioned earlier.

Let's say you wanted to have SAP API Management "do something" when calls come in to the /reports or /forecasts resources. At this point you're not telling SAP API Management what to do, just that it should be listening for calls to those resources. You do this with conditions. In your SAP API Management API proxy, you can create conditional flows for /reports and /forecasts. For conceptual purposes, the following API proxy XML shows what those conditions might look like.

```
<Flows>
    <Flow name="reports">
        <Description/>
        <Request/>
        <Response/>
        <Condition>(proxy.pathsuffix MatchesPath &quot;/reports&quot;) and
(request.verb = &quot;GET&quot;)</Condition>
    </Flow>
    <Flow name="forecasts">
        <Description/>
        <Request/>
        <Response/>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        <Condition>(proxy.pathsuffix MatchesPath &quot;/forecasts&quot;) and
(request.verb = &quot;GET&quot;)</Condition>
    </Flow>
</Flows>
```

Those conditions say, "When a GET request comes in with /reports and /forecasts in the URL, SAP API Management will do whatever you (the API developer) tell it to, through the policies you attach to those flows.

Now here's an example of telling SAP API Management what to do when a condition is met. In the following API proxy XML, when a GET request is sent to https://<host>:<port>/mygreatweatherforecast/reports, SAP API Management executes the "XML-to-JSON-1" policy in the response.

```
<Flows>
    <Flow name="reports">
        <Description/>
        <Request/>
        <Response>
            <Step>
                <Name>XML-to-JSON-1</Name>
            </Step>
        </Response>
        <Condition>(proxy.pathsuffix MatchesPath &quot;/reports&quot;) and
(request.verb = &quot;GET&quot;)</Condition>
</Flow>
```

3.10.4.2 Note about default flows

In additional to those optional conditional flows, each API proxy also comes with two default flows: a <PreFlow> executed before your conditional flows, and a <PostFlow> executed after your conditional flows. Those are useful for executing policies when any call is made to an API proxy. For example, if you want to verify an app's API key with every call, regardless of the backend resource being accessed, you could put a Verify API Key policy on the <PreFlow>.

For more information, see Flows Configuration.

3.10.4.3 Creating conditional flows to backend resources

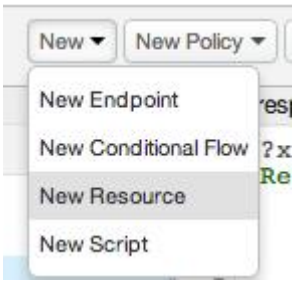
Defining conditional flows to backend resources in an API proxy is completely optional. However, those conditional flows give you the ability to apply fine-grained management and monitoring.

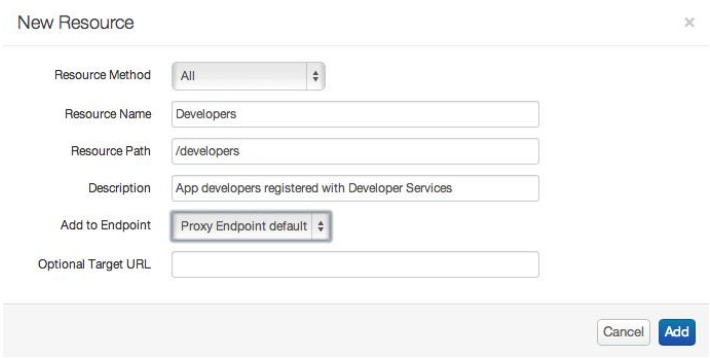
You will be able to:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- Apply management in way that reflects the semantics of your API model
- Apply policies and scripted behavior to individual resource paths (URIs)
- Collect fine-grained metrics for Analytics Services

For example, imagine that you need to apply different types of management to /developers than to /apps. To do so, you add two API resources: /developers and /apps.
In the Develop view of the API proxy builder, select New > Resource.





In the Navigator menu, you can see that two flows have been created: Apps and Developers.

Select one of the flows to view the API resource configuration:

```
<Flow name="Apps">
    <Description>Apps registered in Developer Services</Description>
    <Request/>
    <Response/>
    <Condition>(proxy.pathsuffix MatchesPath &quot;/apps&quot;)</Condition>
</Flow>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

As you can see, API resources are simply conditional Flows that evaluate the URI path of the inbound request. (The proxy.pathsuffix variable identifies the URI of the request that follows the BasePath configured in the ProxyEndpoint configuration.)

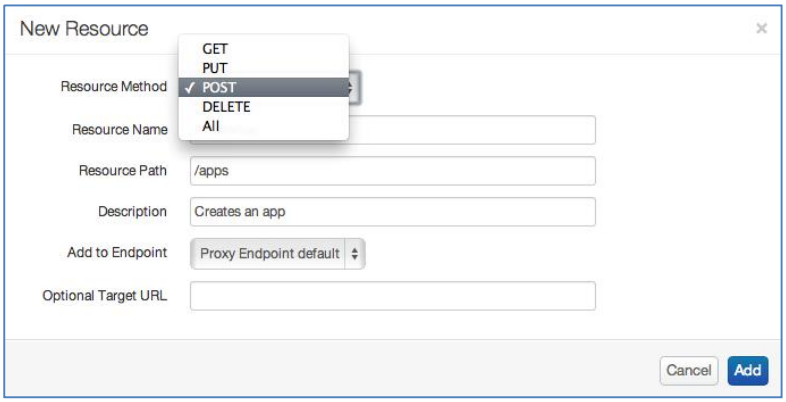
Each API resource that you define is implemented by a conditional Flow in the API proxy. Once you deploy the API proxy to the test environment, the following request:

```
http://{org-name}-{env}.<host:port>/{proxy_path}/apps
```

will cause the condition to evaluate to true, and this flow, along with any associated policies, will execute.

You can further refine resource definitions by specifying the HTTP verb associated with a call:

For example you may need to treat the "create app" method differently than "list apps". To do so, specify the HTTP verb associated with the API resource. In this example, you need to manage the Create App method, so select the POST method.



Select Add.

Adding this API resource results in a new flow. The new flow is added to the ProxyEndpoint for the API proxy that you are building. If you look at the ProxyEndpoint configuration, you will see that the following flow configuration has been added:

```
<Flow name="CreateApp">
    <Description>Creates an app</Description>
    <Request/>
    <Response/>
    <Condition>(proxy.pathsuffix MatchesPath &quot;/apps&quot;) and (request.verb =
&quot;POST&quot;)</Condition>
</Flow>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.4.4 Building conditions

For example, the following condition uses a Java regular expression to recognize calls made to the /apps resource with or without a trailing forward slash (/apps or /apps/**):

```
<Condition>(proxy.pathsuffix JavaRegex &quot;/apps(/?)&quot;) and (request.verb = &quot;POST&quot;)</Condition>
```

3.10.4.5 Modeling Hierarchical URIs

In some cases, you will have hierarchical API resources. For example, the Developer Services API provides a method for listing all apps that belong to a developer. The URI path is:

```
/developers/{developer_email}/apps
```

You may have resources where a unique ID is generated for each entity in a collection, which is sometimes annotated as follows:

```
/genus/:id/species
```

This path applies equally to the following two URIs:

```
/genus/18904/species
```

```
/genus/17908/species
```

To represent this structure in an API resource, you can use wildcards. For example:

```
/developers/*/apps
```

And

```
/genus/*/species
```

will resolve these hierarchical URIs as API resources appropriately.

In some cases, especially for deeply hierarchical APIs, you may simply want to resolve everything below a certain URI fragment. To do so, use a double asterisk wildcard in your resource definition. For example, if you define the following API resource:

```
/developers/**
```

That API resource will resolve the following URI paths:

```
/developers/{developer_email}/apps
```

```
/developers/{developer_email}/keys
```

```
/developers/{developer_email}/apps/{app_id}/keys
```

The benefit of defining API resources is that you gain the ability to apply policies to requests that invoke those specific URIs, providing granular control over the data and services that your API proxy exposes. Additionally, SAP API Management collects operation metrics specific to the API resources you define. By defining specific API resources, you gain the visibility required to identify performance bottlenecks or error conditions as they affect specific calls against your API.

Resources can be used to control access to specific assets or objects in your API. If you disable an API resource, or if you add a security policy to it, you are effectively blocking all the apps that call that resource.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.5 Introduction to Flow Variables

What you'll learn in this topic

This overview topic discusses how to access and use flow variables in SAP API Management API proxies. After reading this topic, you'll understand:

- What flow variables are
- When and where flow variables are created
- How to reference flow variables
- Flow variable scopes and data types
- How to access flow variables in policies
- How to access flow variables JavaScript and Node.js code
- How to create custom flow variables

3.10.5.1 What are flow variables?

Flow variables are named references that hold state associated with an API transaction processed by SAP API Management. They exist within the context of an API proxy flow, and they track state in an API transaction the way named variables track state in a software program. Flow variables store information such as:

- The IP address, headers, URL path, and payload sent from the requesting app
- System information such as the date and time when SAP API Management receives a request
- Data derived when a policy executes. For example after a policy executes that validates an OAuth token, SAP API Management creates flow variables that hold information like the name of the requesting application.
- Information about the response from the target system

Some variables are "built-in" to SAP API Management and are automatically populated whenever an API request is received. They are available throughout an API transaction. You can also create your own custom variables using policies like AssignMessage or in JavaScript, Node.js, and Java code. As you'll see, variables have scope, and where they are accessible depends in part on when they are created in the API proxy flow. In general, when a variable is created, it is available to policies and code that execute later in the API transaction flow.

3.10.5.2 How are flow variables used?

Flow variable are used in policies and conditional flows. Policies can retrieve state from flow variables and use them to do their work. For example, a VerifyAccessToken policy can retrieve the token to be verified from a flow variable and then perform the verification on it. As another example, a JavaScript callout can retrieve flow variables and encode the data contained within those variables. Conditional flows can reference flow variables to direct the flow of an API through SAP API Management, kind of like the way a switch statement works in

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

programming. For example, a policy to return a fault may execute only when a particular flow variable is set. Finally, you can get and set flow variables in a Node.js target application.

Let's look at examples of how variables are used in each of these contexts.

3.10.5.2.1 Flow variables in policies

Some policies take flow variables as input. For example, the following AssignMessage policy takes the value of the flow variable client.ip and puts it in a request header called My-Client-IP. If added to the request flow, this policy sets a header that is passed to the backend target. If set on the response flow, the header would be sent back to the client app.

```
<AssignMessage name="set-ip-in-header">
    <AssignTo createNew="false" transport="http" type="request">request</AssignTo>
    <Set>
        <Headers>
            <Header name="My-Client-IP">{client.ip}</Header>
        </Headers>
    </Set>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</AssignMessage>
```

For another example, when a Quota policy executes, several flow variables are populated with policy-related values. One of these variables is called `ratelimit.my-quota-policy.used.count` (where `my-quota-policy` is the name of the quota policy you're interested in). You might later execute a conditional flow that says "if the current quota count is below 50% of maximum, and it's between 9 am and 5 pm, enforce a different quota." This condition might depend on the value of the current quota count and on a flow variable called system.time, which is one of the built-in SAP API Management variables.

3.10.5.2.2 Flow variables in conditional flows

Conditional flows evaluate flow variables and enable proxies to behave dynamically. Conditions are typically used to change the behavior of flows, steps, and route rules. Here's a conditional flow that evaluates the value of the variable request.verb in a proxy flow step. In this case, if the request verb is POST, the VerifyApiKey policy is executed. This is a common pattern used in API proxy configurations.

```
<PreFlow name="PreFlow">
    <Request>
        <Step>
            <Condition>request.verb equals "POST"</Condition>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
            <Name>VerifyApiKey</Name>
        </Step>
    </Request>
</PreFlow>
```

Now, you might wonder, where do variables like `request.verb`, `client.ip`, and `system.time` come from? When are they instantiated and populated with a value? To help you understand when variables are created and when they're available to you, for more information, see Understanding flow variable scope.

3.10.5.2.3 Flow variables in JavaScript code called with the JavaScript policy

Using the JavaScript policy, you can execute JavaScript code from within the context of an API proxy flow. JavaScript that is executed by this policy uses the SAP API Management JavaScript Object Model, which provides your custom code access to request, response, and context objects associated with the API proxy flow in which your code is executing. For example, this code sets a response header with the value obtained from the flow variable target.name.

```
context.setVariable("response.header.X-SAP-Target",
context.getVariable("target.name"));
```

This techniqe of using JavaScript to read and set variables is similar to the work you can do with the AssignMessage policy (shown previously). It's just another way to accomplish the same kinds of things on SAP API Management. The key to remember is that JavaScript executed by the JavaScript policy has access to all the flow variables that are exist and are in scope within the API proxy flow.

3.10.5.2.4 Flow variables in Node.js code

By requiring the SAP API Management-access module, you can set and access flow variables from within Node.js code that is deployed to SAP API Management.

Here's a simple example where a variable called `custom.foo` is set to the value Bar. Once set, this new variable becomes available to any policies or other code that occurs in the proxy flow after the Node.js code executes.

```
var http = require('http');
var sap = require('sap-access');

http.createServer(function (request, response) {
  sap.setVariable(request, "custom.foo", "Bar");
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Hello World\n');
}).listen(8124);
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

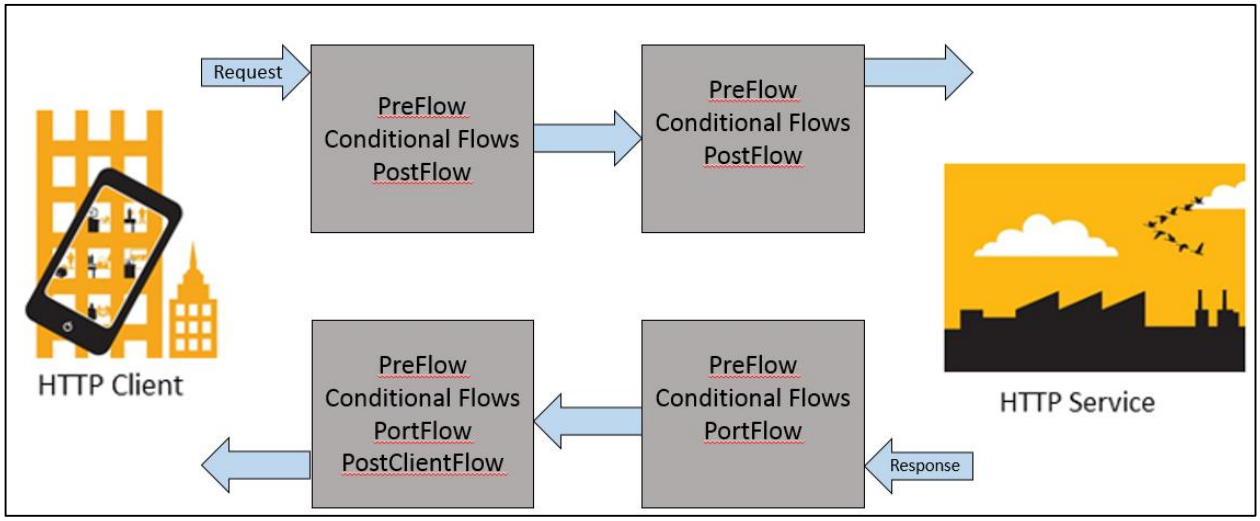
```
console.log('Server running at http://127.0.0.1:8124/');
```

3.10.5.3 Understanding flow variable scope

Variable scope is related to the flow or overall "life cycle" of an API proxy call.

3.10.5.3.1 Visualizing the flow of an API proxy

To understand flow variable scope, it's important to understand or visualize the way messages flow through an API proxy. An API proxy consists of a series or message processing steps organized as a flow. At every step in a proxy flow, the proxy evaluates information available to it and decides what to do next. Along the way, the proxy may execute policy code or perform conditional branching.



Keep this flow structure in mind as we begin to explore flow variables through the rest of this topic.

3.10.5.3.2 How variable scope is related to proxy flow

As soon as you can visualize how messages flow through a proxy, as described previously, you can begin to understand variable scope. By scope, we mean the point in the proxy flow life cycle when a variable is first instantiated. For example, if you have a policy attached to the ProxyEndpoint request segment, that policy will not be able to access any variables that are scoped to the TargetEndpoint request segment. The reason for this is that the TargetEndpoint request segment of the flow has not executed yet, so the API proxy hasn't had a chance to populate variables in that scope.

The following table lists the complete set of variable scopes and indicates when in the proxy flow they become available.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

> **Note**
>
> After a flow variable is populated within its given scope, it remains available through the rest of the proxy flow life cycle. If you try to access a variable that is not yet in scope, you will receive an error or a NULL value.

Variable scope	Where these variables are populated
proxy request	The ProxyEndpoint request segment
target request	The TargetEndpoint request segment
target response	The TargetEndpoint response segment
proxy response	The ProxyEndpoint response segment
always available	As soon as the proxy receives a request. These variables are available through the entire proxy flow life cycle.

For example, there's a built-in SAP API Management variable called `client.ip`. This variable has "proxy request" scope. It is automatically populated with the IP address of the client that called the proxy. It is populated when a request first hits the ProxyEndpoint and remains available through the entire proxy flow life cycle.

There's another built-in variable called `target.url`. This variable's scope is "target request". It is populated in the TargetEndpoint request segment with the request URL sent to the back-end target. If you try to access `target.url` in the ProxyEndpoint request segment, you will receive a NULL value. If you try to set this variable before it is in scope, the proxy does nothing—does not generate an error and does not set the variable.

> **Note**
>
> It's possible to configure some policies to either throw an error or return NULL when an undefined variable is accessed. Refer to individual policy reference pages for details.

Here's a simple example that demonstrates how to think about variable scope. Suppose you want to copy the entire contents of a request object (headers, parameters, body) and assign it to the response payload to be sent back to the calling app. You can use the AssignMessage policy for this task. The policy code looks like this:

```
<AssignMessage name="CopyRequestToResponse">
    <AssignTo type="response" createNew="false">response</AssignTo>
    <Copy source="request"/>
</AssignMessage>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

This policy simply copies the `request` object and assigns it to the response object. But where should this policy be placed in the proxy flow? The answer is that it must be placed on the TargetEndpoint response, because the scope of the response variable is "target response."

3.10.5.3.3 Referencing flow variables

All built-in variables in SAP API Management follow a dot-notation naming convention. This convention makes it easier to determine the purpose of the variable. For example `system.time.hour` and `request.content`.

> ### Note
>
> This dot-notation convention is strictly for naming purposes. For example, `request.content` does not refer to a parent variable or object called "request". You can name custom variables anything you wish. There is no requirement to include a dot. SAP API Management considers following this naming convention a best practice whenever you create your own custom flow variables. It can be useful to use the dots to organize your custom variables. For example `approval.code` and `approval.status` both might refer to an approval feature within the flow. SAP API Management uses prefixes like request, target, system, and response to organize relevant variables appropriately.

When you reference a variable in a policy, you must enclose it in curly braces. For example, this simple AssignMessage policy takes the value of the variable `client.ip` and puts it in a request header called `Client-IP`.

```
<AssignMessage name="set-ip-in-header">
    <AssignTo createNew="false" transport="http" type="request">request</AssignTo>
    <Set>
        <Headers>
            <Header name="Client-IP">{client.ip}</Header>
        </Headers>
    </Set>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</AssignMessage>
```

In conditional flows, the curly braces are not necessary. For example, this condition evaluates the variable `request.header.accept`.

```
<Step>
    <Condition>request.header.accept = "application/json"</Condition>
    <Name>XMLToJSON</Name>
</Step>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.5.3.4 Data type of flow variables

Flow variables have a well-defined data type, such as string, long, integer, boolean, and collection. You can find the data types listed for each built-in SAP API Management flow variable in the Variables reference. For policy-created variables, refer to the specific policy reference topic for data type information. Finally, variables you create manually assume the type given when they were created, and depend on the types of values that are allowed. For example, variables created in Node.js code are restricted to number, String, boolean, null, or undefined.

3.10.5.3.5 Using flow variables in policies

Many policies create flow variables as part of their normal execution. The Policy reference documents all of these policy-specific variables. As you work with proxies and policies, be sure to consult the Policy reference to find out which variables are created and what they are used for. For example, the Quota policy creates a set of variables, that contain information about quota counts and limits, expiry time, and so on. See the section "Policy-specific variables" in Quota policy for the complete list. Some policy variables are useful for debugging. You can use the Trace tool, for instance, to see which variables were set at a particular instance in a proxy flow.

The ExtractVariables policy lets you populate custom variables with data extracted from messages. You can extract query parameters, headers, and other data. For example, you can parse request and response messages using patterns to extract specific data from the messages.

In this example, ExtractVariables is used to parse a response message and store specific data taken from the response. Two custom variables are created and assigned values. The variables are called `geocoderesponse.latitude and geocoderesponse.longitude`.

```
<ExtractVariables name="ParseGeocodingResponse">
  <Source>response</Source>
  <VariablePrefix>geocoderesponse</VariablePrefix>
  <JSONPayload>
    <Variable name="latitude">
      <JSONPath>$.results[0].geometry.location.lat</JSONPath>
    </Variable>
    <Variable name="longitude">
      <JSONPath>$.results[0].geometry.location.lng</JSONPath>
    </Variable>
  </JSONPayload>
</ExtractVariables>
```

Again, be aware that many policies automatically create variables. You can access those variables within the proxy flow context, and they are documented in the Policy reference under each individual policy topic.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.5.3.6 Working with flow variables in JavaScript code

You can access and set variables directly in JavaScript code that is executing in the context of an API proxy. Through the SAP API Management JavaScript Object Model, JavaScript executing on SAP API Management has direct access to proxy flow variables.

To access variables in JavaScript code, call getter/setter methods on any of these objects:

- `proxyRequest`
- `targetRequest`
- `targetResponse`
- `proxyResponse`
- `context`

As you can see, these object references map to the familiar segments of the proxy flow model as explained previously in "Visualizing the flow of an API proxy."

The context object corresponds to "globally" available variables, such as system variables. For example, you can call getVariable() on the `contextobject` to get the current year:

```
var year = context.getVariable('system.time.year');
```

Similarly, you can call setVariable() to set the value of a custom variable or for any writable out-of-the-box variables. Here, we create a custom variable called organization.name.myorg and assign a value to it.

```
var org = context.setVariable('organization.name.myorg', value);
```

Because this variable is created with the context object, it will be available to all flow segments (essentially, this is like creating a global variable).

You can also get/set proxy flow variables in Java code that you execute with the JavaCallout policy.

3.10.5.3.7 Accessing flow variables in Node.js applications

You can get, set, and delete flow variables from Node.js code deployed to SAP API Management. All you need to do is "require" the SAP API Management-access module in your code.

What you need to remember

Here are a few important things to remember about flow variables:

- Some "out-of-the-box" variables are instantiated and populated automatically by the proxy itself. These are documented in Understanding flow variable scope.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- You can create custom variables that are available for use in the proxy flow. It's possible to create variables using policies like AssignMessage and JavaScript, and in Node.js code.
- Variables have scope. For example, some variables are automatically populated when the first proxy receives a request from an app. Other variables are populated in the response flow segment of the proxy. These response variables remain undefined until the response segment executes.
- When policies execute, they can create and populate policy-specific variables. The documentation for each policy lists all of these relevant policy-specific variables.
- Conditional flows typically evaluate one or more variables. You need to understand variables if you want to create conditional flows.
- Many policies use variables as input or output. Perhaps a variable that is created by one policy is later used by another.
- You can get and set many flow variables from within Node.js, straight JavaScript, or Java code executing on SAP API Management.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.6 Reusable Shared Flows

You can combine policies and resources into a shared flow that you can consume from multiple API proxies, and even from other shared flows. Although it's like a proxy, a shared flow has no endpoint. It can be used only from an API proxy or shared flow that's in the same organization as the shared flow itself.

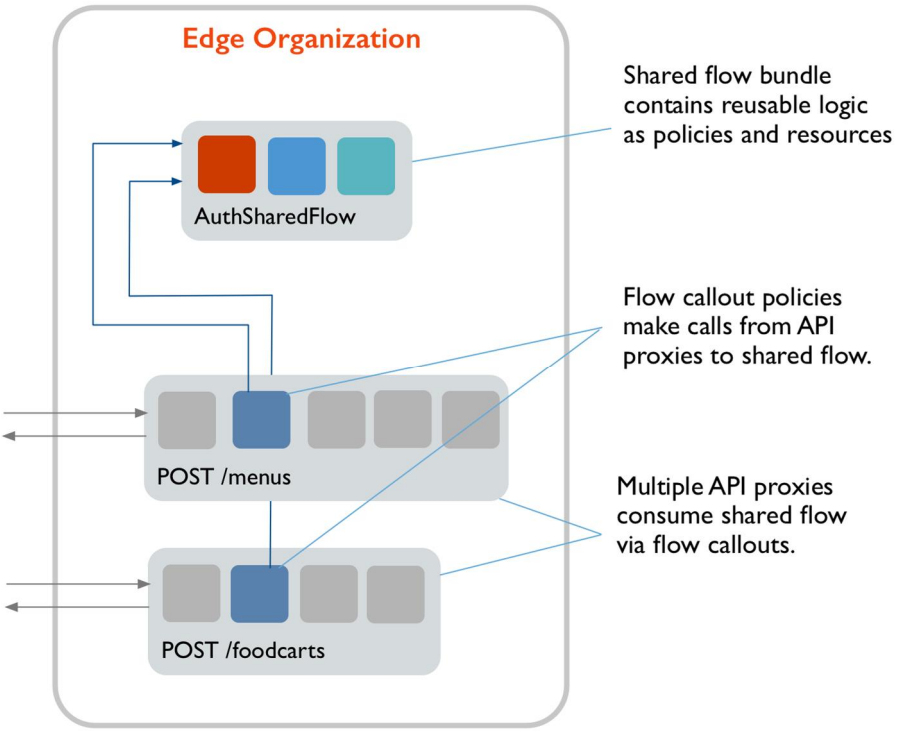
By capturing in one place functionality that's useful in multiple places, a shared flow helps you ensure consistency, shorten development time, and more easily manage code.

You can call a shared flow using the Flow Callout policy. Also, by attaching a shared flow to a flow hook, you can have the shared flow execute before a proxy or target request, or after a proxy or target response.

For example, imagine you have areas of functionality that are either used in multiple places or must be standardized across APIs in your organization. You could have a shared flow for each category, including:

- security, with authorization code using OAuth and API key verification, as well as threat protection code.
- logging, for generating standard error messages.
- mediation, for transforming between XML and JSON message formats.

In the following illustration, two API proxies call out (with a FlowCallout policy) to a shared flow to authenticate incoming user requests. The AuthSharedFlow has been deployed separately to the org before the proxies so that it is available to support requests from the proxies. A shared flow can be developed and managed by a team responsible for broad company policies, then consumed in proxies by line-of-business teams building more specialized apps.



CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.6.1 Developing a shared flow

When developing a shared flow, you must always test it with calls sent to an API proxy. In other words, you can't send requests directly to a shared flow as you would an API proxy. Instead, you send requests to an API proxy which is, in turn, calling out to the shared flow.

Here are the high-level steps for developing a shared flow:

1. Figure out what the shared set of features should be.

   For example, you might want to combine traffic management features, including suppressing traffic spikes. That way, you can manage their configuration outside the workflow of those implementing line-of-business logic.

2. Develop a shared flow by implementing policies and supporting resources, just as you would when developing an API proxy.

   A shared flow is a sequence of conditional steps. So developing one is like developing an API proxy. You can include policies and resources you might include in a proxy.

   For example, as part of your traffic management support, you might implement a Spike Arrest policy to allow only 30 requests per second, as in the following example:

   ```
   <SpikeArrest async="false" continueOnError="false" enabled="true" name="Spike-Arrest">
    <DisplayName>Spike Arrest</DisplayName>
    <Properties/>
    <Identifier ref="request.header.some-header-name"/>
    <MessageWeight ref="request.header.weight"/>
    <Rate>30ps</Rate>
    </SpikeArrest>
   ```

   Then, to a shared flow for traffic management, you could attach the Spike Arrest policy as a step. The policy would execute for any API proxy that calls the shared flow.

   ```
   <SharedFlow name="default">
       <Step>
           <Name>Spike-Arrest</Name>
       </Step>
   </SharedFlow>
   ```

   As with API proxies, you can import a zip file containing your shared flow source artifacts (see Import a new API Proxy for more about importing proxies). The following illustrates how to import a shared flow with the management API:

   ```
   curl -X POST -F "file=@/path/to/zip/file.zip" \
   'https://api.enterprise.apigee.com/v1/o/{org_name}/sharedflows?action=import&name=shared-flow-name' \
   -u email:password
   ```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3. Deploy the shared flow to the environment before deploying proxies or shared flows that will consume it. You deploy a shared flow in the same way you deploy an API proxy.

   A shared flow must be in the same org and deployed to the same environment as the API proxies and other shared flows that consume it. Deploying the shared flow before the proxies makes it possible to resolve the proxy's dependency on the shared flow at deploy time.

   You can deploy a shared flow with a management API call such as the following:

   ```
   curl -X POST --header "Content-Type: application/octet-stream" \
   https://api.enterprise.apigee.com/v1/o/{org_name}/e/{env_name}/sharedflows/{shared_flow_name}/revisions/{revision_number}/deployments \
   -u email:password
   ```

   You can also replace a currently deployed shared flow with no downtime.

   Here's the request form using the management API:

   ```
   curl -X POST --header "Content-Type:application/x-www-form-urlencoded" \
   https://api.enterprise.apigee.com/v1/o/{org_name}/e/{env_name}/sharedflows/{shared_flow_name}/revisions/{revision_number}/deployments?"override=true" \
   -u email:password
   ```

4. Develop the consuming API proxy so that it can call the shared flow as part of its own flow.

   From an API proxy, you call to a shared flow with a Flow Callout policy.

   Note: To consume a shared flow, an API proxy must be deployed after the shared flow and to the same environment as the shared flow. The shared flow has to be in place when you want to test a call to it using the Flow Callout policy.

   To consume a shared flow, you add a Flow Callout policy to the proxy or shared flow that will consume it. Like a Service Callout policy, with which you call out to another service, a Flow Callout calls out to the shared flow.

   In the following code, a Flow Callout policy calls out to a shared flow called traffic-management-shared.

   ```
   <FlowCallout async="false" continueOnError="false" enabled="true" name="Traffic-Management-Flow-Callout">
       <DisplayName>Traffic Management Flow Callout</DisplayName>
       <Properties/>
       <SharedFlowBundle>traffic-management-shared</SharedFlowBundle>
   </FlowCallout>
   ```

5. Deploy the consuming API proxy to begin using the shared flow.
6. Develop iteratively by tracing, as you would with an API proxy.

   As with an API proxy, you develop a shared flow by iteratively invoking and tracing until you have the logic the way you want it. In this case, because the shared flow doesn't run on its own, you invoke a proxy endpoint and trace the proxy.

   Here are the steps:

   1. Make sure that both the shared flow and the API proxy calling it with a Flow Callout policy are in the same organization and deployed to the same environment.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

2. On the API proxy's Trace tab, begin tracing the API proxy.
3. Send a request to a proxy endpoint in the API proxy. The flow from the endpoint must include the Flow Callout policy that calls out to the shared flow.
4. On the Trace tab, examine the flow from the API proxy to the shared flow.

   Notice that in tracing, the shared flow is represented as a set of steps or policies enclosed in a grey box. Icons representing Flow Callout policies precede shared flows.

3.10.6.2 Creating a shared flow with the SAP API Management Edge management console

When you use the SAP API Management Edge UI to create a shared flow, you can either create one from scratch or by importing existing flow sources as a flow bundle .zip file.

1. In the management console, select the organization within which you want to create the shared flow.

   The shared flow will be available to any API proxies and shared flows deployed to an environment from this organization. It won't be available from outside this organization.
2. Click the APIs > Shared Flows menu.

   On the Shared Flows page, you can view a list of shared flows in the organization. You can edit or delete flows in the list.
3. Click the + Shared Flow button to begin adding a new shared flow.
4. On the Build a Shared Flow page, choose how you want to created the new flow:
   - Create a new flow from scratch. You'll be able to configure policies and resources as steps in the flow.
     1. Select Empty Shared Flow.
     2. Enter a name value. This will be the name that API proxies and other shared flows use to reference this shared flow. The name should be descriptive to developers consuming the flow.
     3. Enter a description to provide more information about what the flow does.
     4. Click Next.
     5. Optionally, select the environments to which you want the new flow deployed.

        For example, if you'll be testing the shared flow from API proxies deployed to the test environment, then deploy the shared flow to test.
     6. Click Build and Deploy to have the new shared flow created and deployed to the environment(s) you selected. If you selected no environment, the shared flow will be created, but not deployed.

   - Create a shared flow from existing sources by uploading a flow bundle.
     1. Select Shared Flow Bundle to specify a .zip file containing the artifacts you want in the new flow. A shared flow bundle contains the source artifacts of a shared flow. For example, if you were to download a shared flow from the management console, you'd have a .zip file with the flow bundle.
     2. Click Next.
     3. Click Choose File to browse for the .zip file containing the shared flow sources you want to import.
     4. In the Shared Flow Name box, enter a name for the imported flow. This will be the name that API proxies and other shared flows use to reference this shared flow. The name should be descriptive to developers consuming the flow.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

5. Click Next.
6. Click Build to build the new flow from the sources you're importing.

3.10.6.3 Calling a shared flow from an API proxy or shared flow

You can call a shared flow from a proxy or from another shared flow by using the Flow Callout policy.

1. In the SAP API Management Edge management console, locate the proxy or shared flow from which you want to call another shared flow.
2. In the Navigator, next to Policies, click the + button.
3. In the list of policies, under Extension, click Flow Callout.
4. Enter the display name and name (unique identifier), then select the shared flow that this policy will call.
5. Click Add.
6. Add the new Flow Callout policy to the proxy where you want the call to be made.

3.10.7 Policy Attachment and Enforcement

SAP API Management enables you to 'program' API behavior without writing any code, by using 'policies'. A policy is like a module that implements a specific, limited management function. Policies are designed to let you add common types of management capabilities to an API easily and reliably. Policies provide features like security, rate-limiting, transformation, and mediation capabilities, saving you from having to code and maintain this functionality on your own.

You're not limited to the set of policy types provided by SAP API Management. You can also write custom scripts and code (such as JavaScript), that extend API proxy functionality and enable you to innovate on top of the basic management capabilities supported by SAP API Management Policies

Policy Types

Technically, a policy is an XML-formatted configuration file. Each policy type's structure (for example, the required and optional configuration elements) is defined by an XML schema. SAP API Management Policy types are grouped into the following functional categories:

Traffic Management

Policies in the traffic management category enable you to control the flow of request and response messages through an API proxy. These policies support both operational and business-level control. They give you control over raw throughput, and can also control traffic on a per-app basis. Traffic management policy types enable you to enforce quotas, and they also help you to mitigate denial of service attacks.

Mediation

Policies in the mediation category enable you to actively manipulate messages as they flow through API proxies. They enable you to transform message formats, from XML to JSON (and vice-versa), or to transform one XML

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

format to another XML format. They also enable you to parse messages, to generate new messages and to change values on outbound messages. Mediation policies also interact with basic services exposed by API Platform, enabling you to retrieve data about apps, developers, security tokens, and API products at runtime.

Security

Policies in the security category support authentication, authorization, as well as content-based security.

Extension

Policies in the extension category enable you to tap into the extensibility of API Platform to implement custom behavior in the programming language of your choice.

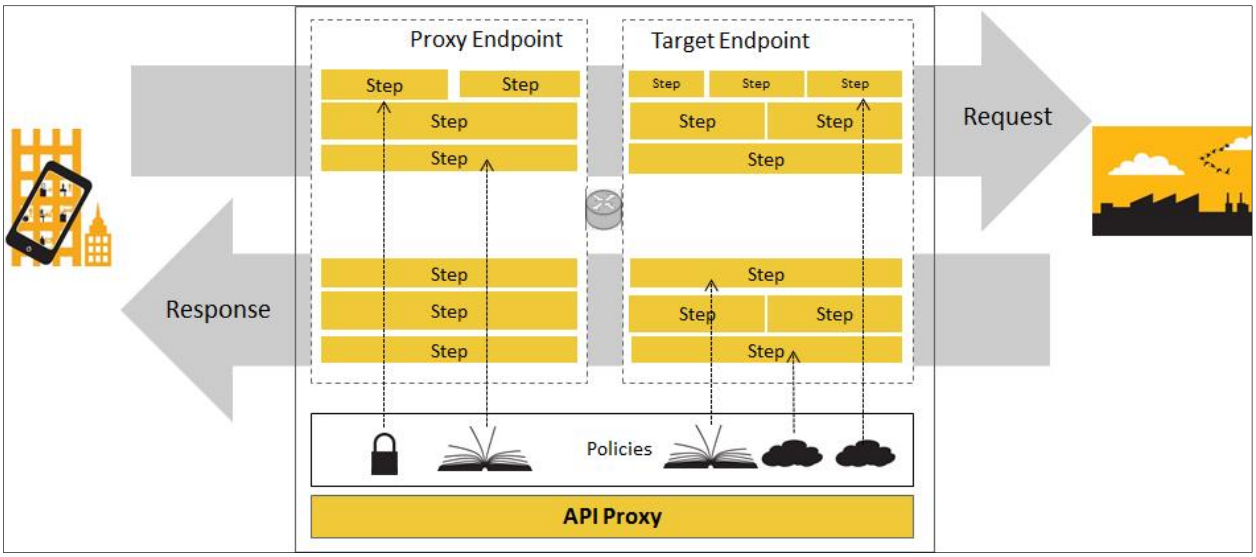
Each Policy type is documented in detail in the Policy Section. This topic demonstrates general interaction, showing you how to create Policies, and how to attach them to Flows in an API proxy configuration.

3.10.7.1 Configuring and Attaching Policies

Adding policy-based capabilities to an API proxy is a two-step process:

1. Configure an instance of a policy type.
2. Attach the policy instance to a Flow.

The diagram below shows the relationship between policies and Flows. As you can see, a policy is attached to a Flow as a processing "Step". To configure the desired behavior for your API, you need to understand a little bit about Flows. (This topic was covered earlier in Flows Configuration.)



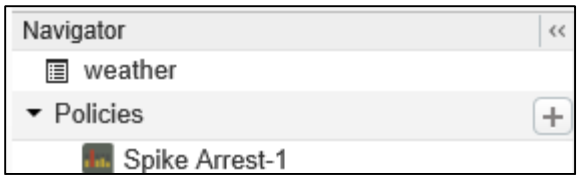
One type of policy that is commonly used is SpikeArrest. SpikeArrest prevents sudden increases in message traffic that might swamp your backend services.

To attach a policy to a Flow:

1. Select an API proxy and navigate to the Develop view.
2. In the API Proxy Editor, select the New Policy drop-down menu.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

A categorized list of the available policy types displays.

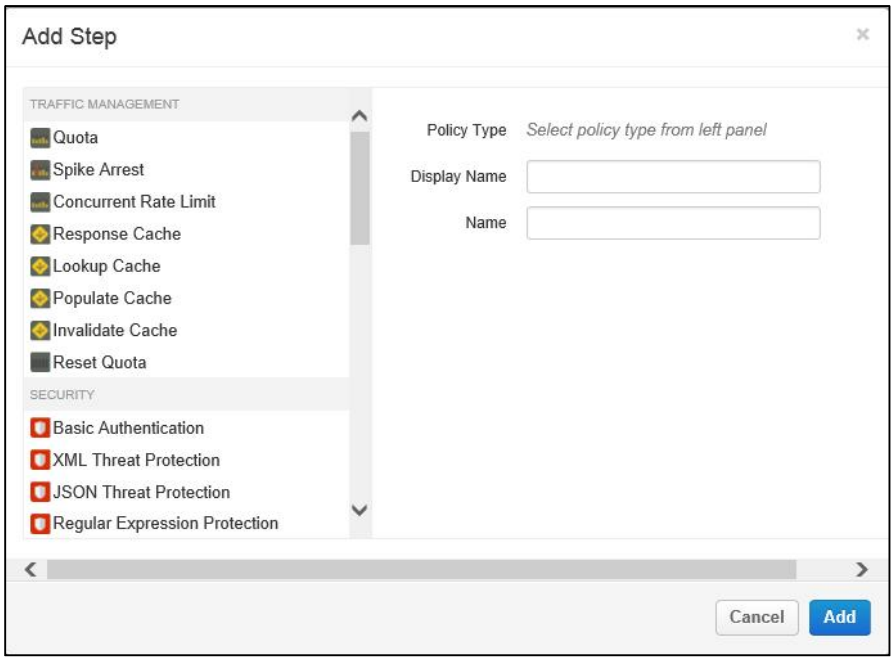


3. Select the policy type that you want to add to the API proxy.

   For example, if you want to add a SpikeArrest policy, select the Traffic Management category in the policy dropdown menu.

4. Modify the selections in the New Policy dialog to configure and attach the policy.

   If you accept the default selections, as shown below, the policy will be enforced on request messages submitted by client apps to the ProxyEndpoint PreFlow.



The New Policy form provides the following options:

- Display Name: A unique name for this policy. The UI will generate a default name, but it is advisable to create a descriptive name for the policy. This will ensure that other developers in your organization have an easy time understanding what the policy is intended to do. The UI changes spaces to dashes, changes consecutive dashes to a single dash, and removes characters that are not alphanumeric, such as dashes, underscores, and percent signs.
- Attach Policy: By selecting this checkbox, you will cause the policy to be attached to the specified Flow when the policy is created, that is, when you select Add. Deselect this box if you do not want to attach the policy yet.
- Flow: Displays a drop-down list of Flows in this API proxy. Select the Flow to which the policy should be attached. Four default Flows are available. (If you added conditional Flows to your API proxy, they will display here as options.) To learn more about Flows, see Flows Configuration.
- Segment: Each Flow has a request and response 'segment'. Select the radio button for the segment to which you want to attach the policy. It's important to attach a policy to the right Flow and segment.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Recommendation

Of course, if you deselect the Attach Policy checkbox, the policy won't be attached to a Flow. The policy will be created, but it won't be enforced. You might do this if you simply want to configure a policy and later decide its attach points. You can attach the policy later by selecting it in the Navigator view and then choosing Attach Policy.)

5. When you finish configuring the policy, select Add

   The policy is attached to the Flow that you selected.

6. After you select Add, you'll see the policy displayed in the Designer view for the PreFlow of the default ProxyEndpoint. The Code view, which displays the XML for the newly attached policy, displays below the Designer view of the Flow. Note that the UI has generated an instance of the policy that contains a set of reasonable default values.

3.10.7.2 Configuring Policies

When the UI generates a policy instance, it applies reasonable default values for common settings. You may need to modify these settings to meet your requirements.

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SpikeArrest async="false" continueOnError="false" enabled="true"
name="spikearrest-1">
  <DisplayName>SpikeArrest-1</DisplayName>
  <FaultRules/>
  <Properties/>
  <Identifier ref="request.header.some-header-name"/>
  <MessageWeight ref="request.header.weight"/>
  <Rate>30ps</Rate>
</SpikeArrest>
```

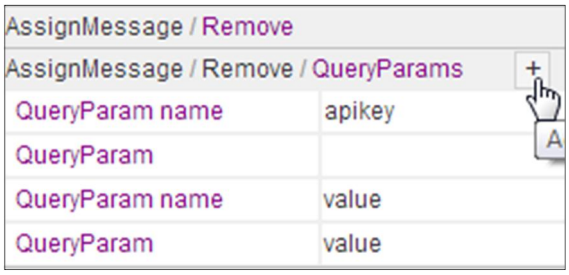
You can configure a policy by directly editing its XML configuration in the Code view. For example, the peak message rate for the Spike Arrest policy is initially set to 30 messages per second. You can change the peak rate by changing the <Rate> element value in the XML for the policy.

You can also cut-and-paste policies into the Code view.

When you make changes to a policy definition in the Code view, the changes are reflected in the Property Inspector. The reverse is also true — make changes in the Property Inspector and they appear in the XML in the Code view.

Some policies expose XML elements that can have multiple subelements. For example, the AssignMessage policy (which is used in the request or response path to generate a new message or to modify an existing message) exposes various elements in which multiple query parameters can be specified. In these cases, the Property Inspector displays a (+) icon. Choose the icon to add entry fields for additional subelements.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.



3.10.7.3 Attaching and Configuring Policies in XML Files

You can create and edit Policies locally, using your favorite text or XML-aware editor or IDE. This topic uses the Quota policy type as an example of how to create, configure, attach, deploy, and test policies.

Most API proxies enforce a quota. Quotas provide control over how often a client app is permitted to invoke an API over a given time interval. In the example below, a Quota policy is configured to limit apps to 1 request per minute. (While this may not be realistic, it does provide a simple way to see the effects of a policy.)

In an API proxy configuration, Policy files are stored as XML files under /apiproxy/policies directory.

For example, a policy of type Quota called "QuotaPolicy" could be created as a file called QuotaPolicy.xml with the following content:

```
<Quota enabled="true" continueOnError="false"
async="false" name="QuotaPolicy">
    <Allow count="1"/>
    <Interval>1</Interval>
    <TimeUnit>minute</TimeUnit>
</Quota>
```

You can create a text file by hand, or you can generate the policy from an XML schema. All policies have some settings that are specific to the policy type, and some settings that are generic across all policies.

When you attach policies in the management UI, the API proxy builder generates the policy instance from the XML schema for the policy type you selected. Therefore, you may see elements in the policy configuration that, for clarity, are not always included in documentation.

All policies define the following attributes:

- enabled: Indicates whether the policy is turned "on" or "off". Policies can be enabled/disabled at runtime by changing this setting. A policy that has enabled set to false is not enforced.
- continueOnError: Defines whether the pipeline should continue processing the message if the policy fails. When enforcing quota policies, errors likely indicate that the quota has been exceeded, and, therefore, this attribute should be set to false.
- async: In a policy, enabling async=true tells API Plarform to run the policy inside a different thread pool, isolated from the regular pool that is servicing the request/response Flow. This is an internal optimization that will rarely be of use to API developers.
- name: The name that you give to this policy. This name is unique to this policy instance, and it is used to attach the policy to the flow as a processing step.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Recommendation

Except for name, you rarely need to modify the default settings for these policy attributes. For this reason, and for clarity, they are often excluded from the policy samples in the documentation.

In the example above, the elements Allow, Interval, and TimeUnit are specific to the Quota policy. These elements provide settings that API Platform enforces on behalf of an API.

3.10.7.4 Attaching a Policy to a ProxyEndpoint or TargetEndpoint Flow

Policies are not executed until they are attached to a Flow. You can create a Policy attachment by naming a Policy in a Step configuration.

The choice of attachment point is critical to the behavior of your API proxy. For example, if you attach the Quota policy to a response Flow, then the Quota would be enforced after the request message was sent to the backend service. That would defeat the purpose of applying a Quota policy! Therefore, you need to attach the Quota policy as a processing Step on the request Flow.

The format of a policy attachment is:

```
<Step>
    <Name>{policy_name}</Name>
</Step>
```

Example

```
<Step>
    <Name>QuotaPolicy</Name>
</Step>
```

Recommendation

The {policy_name} variable must be the same as the name attribute in the policy (stored under /policies) that you want to attach. The names must match exactly, as name matching is case-sensitive.

A policy is attached to a Flow by adding the Step configuration to the appropriate request or response Flow element in a ProxyEndpoint or TargetEndpoint configuration.

You can attach a policy to a request or response Flow. Request and response Flows are further subdivided in to PreFlow and PostFlow.

The following example demonstrates the minimal ProxyEndpoint configuration, with no policy attachments. It simply defines the (inbound) HTTPProxyConnection and a RouteRule.

```
<ProxyEndpoint name="default">
    <HTTPProxyConnection>
        <BasePath>/weather</BasePath>
        <VirtualHost>default</VirtualHost>
    </HTTPProxyConnection>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  <RouteRule name="default">
     <TargetEndpoint>default</TargetEndpoint>
  </RouteRule>
</ProxyEndpoint>
```

You must modify this configuration so that the ProxyEndpoint enforces a Quota policy (as a processing Step) before the API proxy performs any other processing. If a developer has exceeded a Quota, you don't want to waste any computational resources on additional requests.

To enforce this configuration, you attach a processing Step to the request PreFlow as follows:

```
<ProxyEndpoint name="default">
  <PreFlow>
    <Request>
      <Step><Name>QuotaPolicy</Name></Step>
    </Request>
  </PreFlow>
  <HTTPProxyConnection>
    <BasePath>/weather</BasePath>
    <VirtualHost>default</VirtualHost>
  </HTTPProxyConnection>
  <RouteRule name="default">
    <TargetEndpoint>default</TargetEndpoint>
  </RouteRule>
</ProxyEndpoint>
```

Sometimes, you require a policy to execute after the ProxyEndpoint has performed all processing. To do so, attach a policy to the PostFlow request path. The following is a sample request PostFlow attachment. This policy would execute on the request message after all of the policies in the PreFlow execute.

```
<PostFlow>
  <Request>
    <Step><Name>QuotaPolicy</Name></Step>
  </Request>
</PostFlow>
```

The following is a sample response PostFlow attachment. This policy would execute on the response message. (The ProxyEndpoint response PostFlow is the final processing phase before the response is returned to the requesting client app.)

```
<PostFlow>
  <Response>
    <Step><Name>QuotaPolicy</Name></Step>
  </Response>
</PostFlow>
```

The rest of the ProxyEndpoint configuration remains the same. The only change is that you added the PreFlow, Request, and Step elements. In the ProxyEndpoint configuration above, the Quota policy executes

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

when the ProxyEndpoint receives a request from a client app, and before any additional processing takes place.

Recommendation

You can also define conditional Flows that execute between the PreFlow and PostFlow phases. This is covered in the next topic, Flow Variables and Conditions.

3.10.7.5 Deploying Policy Changes

For policy changes to take effect, you must deploy the API proxy revision to an environment. After you attach a policy or make changes to an existing policy, use the management UI or the management API to deploy the changes.

3.10.7.6 Verifying Policy Enforcement

To verify that a policy is enforced properly, the API must be invoked by an HTTP client. To verify this Quota configuration, submit multiple requests to the API, exceeding the quota limit that you set in the quota policy. (The URI path, configured as the base path setting in the ProxyEndpoint, in the request below is /weather).

```
http://<host:port>/weather/forecastrss?w=12797282
```

After you submit more than 1 request within a minute, you should see the following error message:

```
{"fault":{"faultstring":"policies.ratelimit.QuotaViolation","detail":{"errorcode":"pol
icies.ratelimit.QuotaViolation"}}}
```

This indicates that the Quota policy is being enforced by API Platform.

3.10.7.7 Policy-Based Fault Handling

Note the format of the error message above. It contains a faultstring property and an errorcode property. In many cases, you need to implement some behavior to handle these errors. For example, you may wish to issue a customized message to a developer whose app has exceeded the Quota.

To do so, you use the FaultRule element on the Step. A FaultRule element can be attached to any policy. The behavior of the FaultRule is defined by the Policy that you attach to the FaultRule.

Example

```
<Step>
 <FaultRules>
  <FaultRule>
    <Name>QuotaLimitAlertPolicy</Name>
    <Condition>(fault.name = ratelimit.QuotaLimitCheck.exceed.count)</Condition>
  <FaultRule>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<FaultRules>
<Name>QuotaPolicy</Name>
</Step>
```

In the example above the Policy named QuotaLimitAlertPolicy would typically be an AssignMessage Policy that generates a custom response message.

For more on fault handling, see Fault handling

3.10.7.8 Best Practices: Common Policy Sets

To meet basic management requirements, API proxies usually enforce the following policies:

Basic API Key Validation

ProxyEndpoint Request Flow:

1. SpikeArrest
2. XMLThreatProtection or JSONThreatProtection
3. API key validation
4. Quota
5. ResponseCache

ProxyEndpoint Response Flow:

1. ResponseCache

Basic Transformation: JSON to XML

Request Flow:

1. SpikeArrest
2. JSONThreatProtection
3. API key validation
4. Quota
5. JSONToXML

Response Flow:

1. XMLToJSON
2. ResponseCache

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.8 Rate-Limiting

3.10.8.1 SpikeArrest

This policy smooths traffic spikes by dividing a limit that you define into intervals of 10 milliseconds. For example, if you define a limit of 100 messages per second, the SpikeArrest policy enforces a limit of around 10 per millisecond. The SpikeArrest limit should be close to capacity calculated for either your backend service or the API proxy itself. The limit should also be configured for shorter time intervals, such as seconds or minutes. This policy should be used to prevent sudden traffic bursts caused by malicious attackers attempting to disrupt a service using a denial-of-service (DOS) attack or by buggy client applications.

See Shield APIs using SpikeArrest section in the Policy document.

3.10.8.2 Quota

This policy enforces consumption limits on client apps by maintaining a distributed 'counter' that tallies incoming requests. The counter can tally API calls for any identifiable entity, including apps, developers, API keys, access tokens, and so on. Usually, API keys are used to identify client apps. This policy is computationally expensive so, for high-traffic APIs, it should be configured for longer time intervals, such as a day or month. This policy should be used to enforce business contracts or SLAs with developers and partners, rather than for operational traffic management.

See Rate limit API traffic using Quota section in the Policy document.

3.10.8.3 Concurrent Rate Limiting

This policy enables traffic management between API Platform and your backend services. Some backend services, such as legacy applications, may have strict limits on the number of simultaneous connections they can support. This Policy enforces a limit on the number of requests that can be sent at any given time from API Platform to your backend service. This number is counted across all of the distributed instances of API Platform that may be calling your backend service. Policy limits and time duration should be configured to match the capacity available for your backend service.

See Throttle backend connections using ConcurrentRatelimit in the Policy document.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.9 Persistence

3.10.9.1 Introduction

API proxies that run on API Platform are stateless. They enforce policies and execute scripts and code, but the environment in which they run is populated with context that is specific to each request/response transaction.

To provide API proxies with access to data over multiple transactions, API Platform provides a persistence layer that enables API proxies to store data. The persistence layer consists of L1 and L2 caches and a NoSQL key/value store. These caches and the data store work together to optimize availability and performance.

As a developer, you can access this persistence layer without knowing the details of its implementation. Instead, you use a set of policies to interact with the persistence layer at runtime. API Platform defines three policy types that enable you to configure persistence for API proxies using configuration instead of code.

SAP API Management supports these persistence scenarios:

- A cache for serving duplicates of stored static response messages
- A general purpose caching mechanism for persisting any arbitrary objects used by your API proxies
- HTTP/1.1 compliant caching
- A highly performant NoSQL key/value store for simple data sets used by your API proxies

3.10.9.2 Response Cache

Response caching is a dedicated application of the API Platform caching mechanism. Response caching is implemented in the ResponseCache policy, so enabling ResponseCaching for your APIs requires no coding on your part.

Not all data is updated in real-time. Services often only update data periodically, for example, weather data updates every 10 minutes, or product catalog prices update every 24 hours.

When ResponseCache is employed in a proxy, SAP API Management also looks at certain HTTP response caching headers and takes appropriate actions according to the directives of those headers. For example, on responses from backend targets, SAP API Management supports the `Cache-Control` header, which can be used to control the maximum age of a cached response, among other directives. For more information, see HTTP response caching.

A consideration for response caching is that responses larger than 256KB are not distributed across the SAP API Management infrastructure. This is done to optimize performance.

3.10.9.3 Cache

Caching is more general purpose than ResponseCache, enabling you to persist any object that your API proxies require over multiple request/response sessions. API Platform provides policies for populating, retrieving, and flushing the cache at runtime. The interaction mechanism that links the API proxies with the underlying cache is variables. You can use policies or code in your API proxies to populate and retrieve cached objects by interacting

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

with the variables that you define in caching policies. API Platform also exposes a RESTful API for managing Cache resources.

Common uses of Cache include temporary storage of:

- Session IDs for session management
- Credentials used for our outbound calls (such as API keys or OAUth access tokens)
- Response content that must be paginated for apps

3.10.9.4 HTTP/1.1 compliant caching

SAP API Management supports a subset of HTTP/1.1 compliant cache headers, including the cache-control header and entity tags. For more information, see HTTP response caching.

3.10.9.5 Key/value Map

If the data you need to store is structured or is long-lived, then use the key/value map.

Examples of situations where you would use the key/value store are:

- A map of IP addresses to country codes
- A list of IP addresses for whitelisting/blacklisting
- A mapping of long URLs to corresponding shortened URLs

API Platform provides a set of policies for interacting with the key/value store at runtime. It also exposes a RESTful API for managing the contents of the key/value store. The API enables you, for example, to perform bulk operations to populate the key/value store with large data sets.

Therefore, key/value maps work better for single entities that have many properties. For example:

```
curl -H "Content-Type:application/json" -X POST -d \
'{
  "entry" : [ {
    "name" : "development",
    "value" : "dev.apifactory.com"
  }, {
    "name" : "production",
    "value" : "prod.apifactory.com" } ],
  "name" : "URLs"
}' \
http://<host:port>/v1/o/{org_name}/keyvaluemaps \
-u myname:mypass
```

The result is a key/value map that can be retrieved as JSON or XML for use at runtime by policies or code that you write.

You can add entries to the key/value map by using the PUT verb. You only need to include the entries to be added:

```
curl -H "Content-Type:application/json" -X PUT -d \
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
'{
  "entry" : [ {
    "name" : "staging",
    "value" : "stage.apifactory.com"
  } ],
  "name" : "URLs"
}' \
http://<host:port>/v1/o/{org_name}/keyvaluemaps \
-u myname:mypass
```

See Persist data using KeyValueMap in the Policy document.

3.10.9.6 Cache Versus Key / Value Map

The PopulateCache policy does not persist cache entries. Entries are in memory until the configured expiration time. You can look up the value only until it expires. One limitation is that you cannot find the list of keys that are currently in a cache.

When using KeyValueMap, the keys are persisted indefinitely. There are APIs available to retrieve the list of keys. There is no expiration time for the keys; you must explicitly delete them.

3.10.10 HTTP Response Caching

3.10.10.1 About HTTP / 1.1 Response Caching

The HTTP / 1.1 specification describes standard caching headers and control mechanisms that support caching along the HTTP request / response chain. These headers and mechanisms provide information about cached resources and help servers determine how to manage cached data. You can refer to the spec itself for detailed information about caching in HTTP, and there are also many blogs, articles, and other resources available on the internet that explains HTTP caching.

The focus of this topic is to explain how SAP API Management processes and handles HTTP / 1.1 headers related to response caching.

Recommendation

SAP API Management currently supports a subset of the HTTP / 1.1 caching headers and directives. Where appropriate, we'll point out features that are not supported.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.10.2 Caching in SAP API Management

SAP API Management provides a persistence layer that allows HTTP responses and arbitrary variables to be stored in caches. You enable and control this persistence layer with policies. SAP API Management provides policies for storing arbitrary variables (the Cache policies) and a policy for controlling HTTP response caching (the ResponseCache policy).

You can read an overview of the SAP API Management persistence layer in Persistence.

> **Recommendation**
>
> SAP API Management Platform does not perform any caching unless one or more of these policies are attached to a proxy.

This topic concerns HTTP response caching and how the HTTP/1.1 caching headers are handled when the ResponseCache policy is employed by a proxy. To learn about adding and configuring the ResponseCache policy, see Reduce latency using ResponseCache.

3.10.10.3 Enabling HTTP/1.1 Response Caching

When the ResponseCache policy is added to a proxy, HTTP responses are cached and retrieved according to how the proxy is configured. The ResponseCache policy includes two flag elements that, when set, allow the policy to consider HTTP cache headers. If you want to use these HTTP cache header features, you need to set these flag elements. They are:

- `UseResponseCacheHeaders`: If set to true, the HTTP response headers are used when setting the "time to live" (TTL) of the response in the cache. The Expires response header is ignored by default unless this flag is set to true. If the max-age or s-maxage directives of the Cache-Control header are set, they take precedence over the Expires header value.
- `UseAcceptHeader`: When this flag is set to true, the Accept headers in the response are used to generate a cache key. The default is false. See "How are cache keys generated".

In addition, SAP API Management looks for and evaluates certain HTTP caching headers. SAP API Management also considers these headers and takes appropriate action based on their directives. In some cases, these HTTP/1.1 cache headers override whatever behavior is specified in the ResponseCache policy. For example, if the Cache-Control header is returned from a backend server, the header's s-maxage directive overrides the corresponding setting in the ResponseCache policy.

SAP API Management supports the following HTTP cache headers on responses received from backend target (origin) servers:

- Cache-Control
- Expires

On client GET requests, SAP API Management supports the The If-Modified-Since header.

SAP API Management also supports Entity Tags (ETags) and their related GET request headers.

- If-Match
- If-None-Match

Finally, SAP API Management provides support for caching and returning compressed responses.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.10.4 The Cache-Control Response Header

This section discusses SAP API Management support for the Cache-Control header.

Recommendation

Although the HTTP/1.1 spec allows Cache-Control headers in both client requests and origin server responses, SAP API Management only supports the Cache-Control header on responses returned from origin servers (backend targets). Origin servers can include both target endpoints defined in an e SAP API Management API proxy and those created using TargetServer API calls.

When SAP API Management detects a Cache-Control response header, it looks at the header's directives and takes appropriate processing steps.

3.10.10.5 About Cache-Control Header Support

SAP API Management supports a subset of Cache-Control response header capabilities defined in the HTTP/1.1 specification. Please note the following:

SAP API Management does not support Cache-Control headers that arrive with inbound client requests.

According to the HTTP specification, Cache-Control can either be public (shared) or private (single user). However, SAP API Management only supports the notion of public caches.

Not all Cache-Control response directives in the HTTP/1.1 specification are supported by SAP API Management. See "Support for Cache-Control response header directives" below for details.

3.10.10.6 Support for Cache-Control Response Header Directives

The following table describes SAP API Management support for HTTP Cache-Control response header directives.

Recommendation

SAP API Management supports subset directives from the HTTP/1.1 specification on responses from origin servers. For more detailed information on these directives, see "Cache-Control" in the HTTP/1.1 specification.

Cache-Control directive	How SAP API Management processes the directive
Public	SAP API Management caches the origin response, even when other directives indicate otherwise. Per the HTTP/1.1 specification, the only exception to this rule is if the response includes an Authorization header.
Private	This directive is not supported by SAP API Management. If this directive is received, the origin response is not cached. Any field names are ignored.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Cache-Control directive	How SAP API Management processes the directive
no-cache	SAP API Management caches the origin response, but it must be revalidated with the origin server before being used to satisfy any subsequent client requests. This rule allows the origin to return a 304 Not Modified response in order to indicate that the response should be returned from cache, thus saving the processing required to return the entire response. If the origin server returns a full response, it replaces the existing cache entry. Any field names specified with this directive are ignored. Note: The HTTP/1.0 header, Pragma: no-cache is treated as equivalent toCache-Control: no-cache.
no-store	Not supported.
no-transform	Not supported.
must-revalidate	Not supported. All cache entries are deleted by SAP API Management as soon as they expire.
proxy-revalidate	Not supported. All cache entries are deleted by SAP API Management as soon as they expire.
max-age	The response is cached for the specified number of seconds. Note: This directive overrides any expiration specified in the Expires header and the ResponseCache policy. See also Reduce latency using ResponseCache.
s-maxage	The response is cached for the specified number of seconds. Note: This directive overrides any expiration specified in the Expires header and the ResponseCache policy. See also Reduce latency using ResponseCache
cache-extension	Not supported.

3.10.10.7 The Expires Response Header

SAP API Management uses the Expires header on origin server responses to determine the time to live (TTL) of a cached entry. As expected, this header specifies a date/time after which a response's cache entry is considered stale. This header allows servers to signal when it's okay to return a cached value based on a time stamp.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Recommendation

When the UseResponseCacheHeaders flag in the ResponseCache policy is set to true, theCache-Control directives max-age and s-maxage take precedence over and override the behavior of the Expires header.

Acceptable date formats for the Expires header are described in the HTTP/1.1 specification. For example:

Expires: Thu, 01 Dec 1994 16:00:00 GMT

For detailed information on HTTP date/time formats, see "Date/Time Formats" in the HTTP/1.1 specification.

For more information on Expires header, see "Header Field Definitions" in the HTTP/1.1 specification.

3.10.10.8 Entity Tag Support

An entity tag (ETag) is an identifier associated with a requested resource. This identifier allows servers to determine if the requested resource and the associated cached resource match or not. Servers can then take appropriate actions, such as re-caching a response if it does not match or returning the cached resource if the ETags match. When a target endpoint sends a response back to SAP API Management with an ETag, SAP API Management caches the ETag along with the response. You can read more about Entity Tags in "Protocol Parameters" in the HTTP/1.1 specification.

3.10.10.9 If-Match Request Header Support

The If-Match request header allows SAP API Management to determine if a cached entity is current or not based on if the ETag in the header matches the ETag that is cached with the requested entity. You can read more about If-Match in "Header Field Definitions" in the HTTP/1.1 specification.

If SAP API Management receives an inbound GET request from a client that includes an If-Match header:

- If the If-Match header specifies one or more ETags, SAP API Management retrieves any unexpired cached entries for the specified resource and compares any strong ETags on those cached entries with those specified in the If-Match header. If a match is found, the cached entry is returned. If not, the request is passed to the origin server.
- If the If-Match header specifies "*", the request is passed on to the origin server to ensure that any origin caching facilities have a chance to process the request.
- Any requests other than GET that specify an If-Match header are passed on to the origin server to ensure that any origin caching facilities have a chance to process the request.
- If a cache entry with the same request URI is found, but it contains only weak ETags, then the entry must be revalidated by the origin server before being returned to the client.
- Any ETags returned by the origin server are returned unchanged to the client.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.10.10 If-None-Match Request Header Support

The If-None-Match header allows SAP API Management to determine if a cached entity is current or not based on if the ETag in the header does not match the ETag that is cached with the requested entity.

If SAP API Management receives an inbound GET request from a client that includes an If-None-Match header:

- If the `If-None-Match` header specifies one or more ETags, SAP API Management retrieves any unexpired cache entries for the specified URI and compares any strong ETags on those cached entries with those specified in the If-None-Match header. If a match is found, SAP API Management returns a 304 Not Modified status. If no match is found, SAP API Management passes the request to the origin server.
- If the `If-None-Match` header specifies "*" and an unexpired cached entry for the requested URI exists, SAP API Management returns a 304 Not Modified status.
- Requests other than a `GET` that include an `If-None-Match` header are passed on to the origin server.
- If a cache entry with the same request URI is found but contains only weak ETags, then the entry must be revalidated by the origin server before SAP API Management returns it to the client.
- If SAP API Management receives an ETag from an origin server, the ETag is always returned unchanged to the client.

3.10.10.11 The If-Modified-Since header

If SAP API Management receives an `If-Modified-Since` header in a GET request, it is passed along to the origin server even if a valid cache entry exists. This ensures that any updates to a resource that did not pass through SAP API Management are accounted for. If the origin server returns a new entity, then SAP API Management replaces the existing cache entry with the new value. If the server returns a 304 Not Modified status, SAP API Management returns the response value if the cached response's Last-Modified header indicates that it has not changed.

3.10.10.12 Handling Compressed and Uncompressed Data

When an incoming request includes the header `Accept-Encoding` with values of `gzip`, `deflate` or `compress`, the origin server responds with compressed data. When subsequent responses come without the `Accept-Encoding` headers, they expect an uncompressed response. SAP API Management's response caching mechanism is capable of sending both compressed and uncompressed responses depending on the incoming headers without going back to the origin server.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.10.13 How Cache Keys are Generated?

When the ResponseCache flag `UseAcceptHeader` is set to true, SAP API Management uses the Accept, Accept-Encoding, Accept-Language and `Accept-Charset` request headers when calculating the cache key. This approach prevents a client from getting a media type they did not ask for. For example, consider if two requests come in from the same URL, where the first request accepts gzip and the second does not. The first request will be cached, and the cached entry will (probably) be a gzipped response. The second request will read the cached value and may then return a gzipped entry to a client that is not capable of reading gzip.

3.10.10.14 Which Response Codes are Cached?

By default, SAP API Management only caches 2xx level responses. Non-2xx responses are not cached.

3.10.11 Fault Handling

Many error conditions can arise while API proxies are servicing requests from apps. For example, API proxies may encounter network issues when communicating with backend services, apps may present expired credentials, request messages may be incorrectly formatted, and so on. In many cases, you will need to handle such errors in a customized fashion. This topic shows you how to set up custom fault handling using the controls provided by API Platform.

3.10.11.1 About Errors

When an API proxy encounters an error, the default behavior of the API proxy is to exit the normal processing pipeline, meaning exit the current flow, and to enter an Error flow. Once the API proxy enters the Error flow, it cannot return processing back to the normal flow pipeline.

By entering the Error flow, the proxy:

- Bypasses any remaining processing steps, meaning any remaining policies, in the current flow
- Bypasses any subsequent flows
- Returns a fault response to the requesting app.

This process results in a raw error message or error codes being returned to the requesting app.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.11.2 Types of errors

An API proxy can encounter the following categories of errors:

- Messaging: Failures arising while driving the message flow, not including policy errors, such as routing failures
- Policy: Failures arising while processing any policies
- Transport: Failures occurring in the transport layer, such as HTTP errors due to protocol level failures or SSL errors
- System: Failures in the underlying system, such as out of memory errors

For more information on these errors, see Fault Taxonomy below.

3.10.11.3 Controlling how a policy reacts to an error

For all errors other than policy errors, an API proxy immediately enters the Error flow. For policy errors, meaning failures arising while processing a policy, the `continueOnError` element of a policy controls the way the API proxy reacts to an error.

By default, the `continueOnError` element is false, meaning that when a fault occurs in a policy, control is directed to the Error flow. For example, the following VerifyAPIKey policy uses the default value of the continueOnError element:

```
<VerifyAPIKey async="false" continueOnError="false" enabled="true" name="verify-api-key">
  <DisplayName>Verify API Key</DisplayName>
  <APIKey ref="request.queryparam.apikey"/>
</VerifyAPIKey>
```

If you set the continueOnError element to true, control stays in the current flow of the API proxy and the next policy in the pipeline executes.

3.10.11.4 Creating fault rules

In many cases you want to control the response of your API proxy to an error. For example, you might want to send a custom error message when a developer app exceeds a Quota, add more information about the error to the response, remove information from the response not meant to be seen by the calling app, or to modify the response in other ways to improve usability and security.

SAP API Management enables you to customize error handling by defining fault rules. A fault rule specifies two items:

- A Condition that specifies the fault to be handled based on the pre-defined category, subcategory, or name of the fault
- One or more policies that define the behavior of the fault rule for the corresponding Condition.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Very typically it will be a single AssignMessage policy that overrides the default message generated for a fault and inserts a customized error message. Usually FaultRules do not and should not themselves use RaiseFault directly. The fault has already been raised. The execution is already in the Error flow. Therefore there is no need to use RaiseFault to raise a new fault.

Attach fault rules to the following entities in an API proxy configuration:

- ProxyEndpoint: Enables fault handling for all errors that occur in the ProxyEndpoint request and response flows.
- TargetEndpoint: Enables fault handling for all errors that occur in the TargetEndpoint request and response flows.

> Hint
>
> To add a fault rule you need to edit the XML configuration of the ProxyEndpoint or TargetEndpoint. You can use the SAP API Management UI to make this edit in the Code pane of the Develop view for an API proxy, or edit the XML file that defines the ProxyEndpoint or TargetEndpoint.

The basic structure of a FaultRule is shown below:

```
<FaultRules>
  <FaultRule>
    <Step>
      <Name>{policy_name}</Name>
    </Step>
    <Condition>{(conditional statement)}</Condition>
  </FaultRule>

  <FaultRule>
    ...
  </FaultRule>
</FaultRules>
```

OR

```
<FaultRules>
  <FaultRule>
    <Step>
      <Name>{policy_name}</Name>
      <Condition>{(conditional statement)}</Condition>
    </Step>
  </FaultRule>

  <FaultRule>
    ...
  </FaultRule>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
</FaultRules>
```

where the Condition applies to the entire fault rule or only to a specific step in the fault rule. You can define multiple fault rules in a ProxyEndpoint or TargetEndpoint to handle multiple fault conditions.

When a FaultRule's condition evaluates to true, then the Policy named in the FaultRule will execute and any subsequent policies with matching conditions will execute. This is different behavior from Route Rules and Flow conditions which stop once a condition is met. In those cases, even if environment conditions match subsequent Route Rules and Flow conditions, those Route Rules and Flows will be ignored.

Recommendation

You can learn how to configure conditions by referring to the topic Flow Variables and Conditions.

3.10.11.5 Specifying the condition on a fault rule

Each type of error defines a set of error codes that it creates in response to an error. For policy errors, you can find the error codes for each policy in the policy reference, starting with the Policy Reference.

All policy error codes have the same format, as shown in the example below:

```
{
  "code" : " {error.code} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

For example, when a developer app presents and invalid consumer key, the VerifyApiKey policy returns the following error code:

```
{
  "code" : " InvalidApiKey ",
  "message" : "The consumer key presented by the app is invalid.",
  "contexts" : [ ]
}
```

Use the error code to define the condition the defines the fault rule to execute. The fault.name variable contains the value of the code field of the error code. You then use the fault.name variable in the <Condition> tag, as the following example shows:

```
<ProxyEndpoint name="default">
  ...
  <FaultRules>
    <FaultRule name="invalid_key_rule">
      <Step>
        <Name>{policy_name}</Name>
      </Step>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
      <Condition>(fault.name = "InvalidApiKey")</Condition>
    </FaultRule>
  </FaultRules>
</ProxyEndpoint>
```

This fault rule executes if an error code InvalidApiKey is thrown by the VerifyApiKey policy.

One common configuration of an API proxy is to use the Quota policy to set the number of requests that an app is allowed to submit to an API proxy over a time period. If the app exceeds the allowed quota, then the Quota policy issues an error where the the value of the code field is `QuotaViolation`. You can use the following condition in a fault rule to detect the quota error:

```
<FaultRule name="quota_rule1">
  <Step>
    <Name>{policy_name}</Name>
  </Step>
  <Condition>(fault.name = "QuotaViolation")</Condition>
</FaultRule>
```

> **Note**
>
> If multiple fault rules have a condition that evaluates to true, then the last of those fault rules executes.

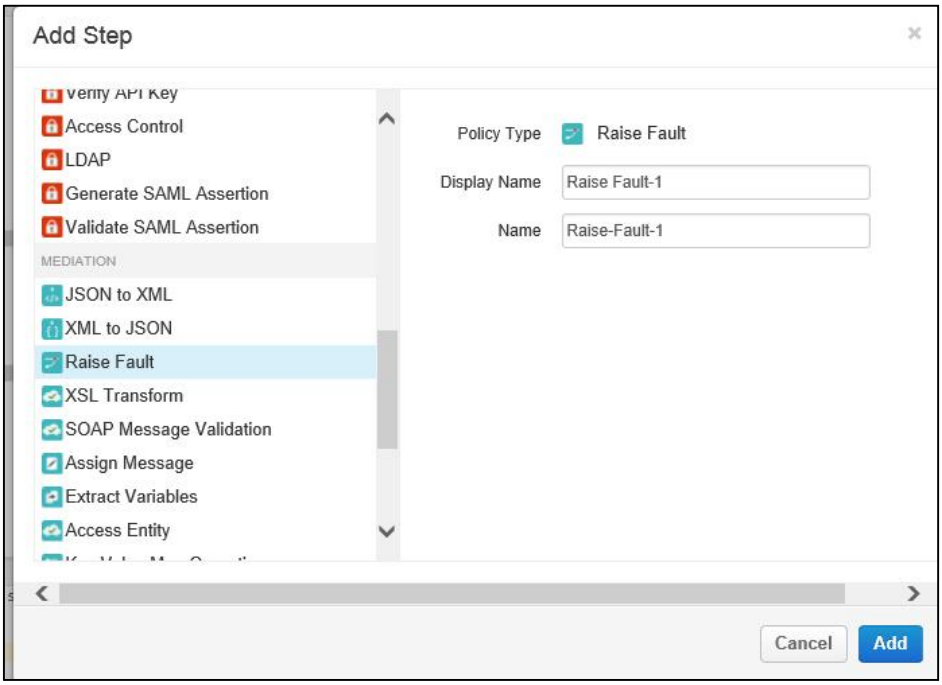
3.10.11.6 Adding policies to a fault rule

Now that you have configured a fault rule, you need to add a policy to the rule to handle the error. This policy then determines the action of the fault rule.

While you can put any policy in the fault rule, you commonly use the Raise Fault policy. Use the RaiseFault policy to generates a custom response message for an error condition. RaiseFault enables you to configure an HTTP response with payload, HTTP status code, headers, and reason phrase elements.

> **Note**
>
> A policy used in a fault rule is typically not used in the normal flow pipeline. However, when using the SAP API Management UI to create a policy, the SAP API Management UI by default assigns the policy to a flow. For example, it can assign it to the Flow PreFlow, ProxyEndpoint default. Therefore, when using the SAP API Management UI to create a policy for a fault rule, you typically uncheck the Attach Policy checkbox, as shown below, so that the policy is not assigned to a flow:

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.



The example below shows a typical RaiseFault policy configuration:

```
<RaiseFault name="fault_invalidkey">
  <FaultResponse>
    <Set>
      <Payload contentType="text/plain">Contact support at
support@mycompany.com.</Payload>
      <StatusCode>401</StatusCode>
      <ReasonPhrase>Unauthorized</ReasonPhrase>
    </Set>
  </FaultResponse>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</RaiseFault>
```

You can now use this policy in your fault rule. Notice how you reference the RaiseFault policy by name in the fault rule:

```
<ProxyEndpoint name="default">
  ...
  <FaultRules>
    <FaultRule name="invalid_key_rule">
      <Step>
        <Name>fault_invalidkey</Name>
      </Step>
      <Condition>(fault.name = "InvalidApiKey")</Condition>
    </FaultRule>
  </FaultRules>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
</ProxyEndpoint>
```

When you deploy the configuration above, the API proxy will execute the `RaiseFault` policy called `fault_invalidkey` whenever an app presents and invalid API key.

You can execute multiple policies in a fault rule, as the following example shows:

```
<ProxyEndpoint name="default">
  ...
  <FaultRules>
    <FaultRule name="invalid_key_rule">
      <Step>
        <Name>policy1</Name>
      </Step>
      <Step>
        <Name>policy2</Name>
      </Step>
      <Step>
        <Name>policy3</Name>
      </Step>
      <Condition>(fault.name = "InvalidApiKey")</Condition>
    </FaultRule>
  </FaultRules>
</ProxyEndpoint>
```

The policies execute in the order defined. For example, you can use the Message Logging policy, the Extract Variables policy, the AssignMessage policy, or any other policy in the fault rule. Note that processing of the fault rule stops immediately if either of these situations occur:

- Any policy in the fault rule causes an error
- Any of the policies in the fault rule is of type RaiseFault

3.10.11.7 Defining the custom error message returned from a fault rule

As a best practice, you should define a rules that determine the error responses from your APIs. In that way, you deliver consistent and helpful information your clients.

The example Raise Fault policy below uses the `<Payload>`, `<StatusCode>`, and `<ReasonPhase>` tags to define the custom error response sent back to the client:

```
<RaiseFault name="fault_invalidkey">
  <FaultResponse>
    <Set>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
      <Payload contentType="text/plain">You have attempted to access a resource
without the correct authorization.
         Contact support at support@mycompany.com.</Payload>
      <StatusCode>401</StatusCode>
      <ReasonPhrase>Unauthorized</ReasonPhrase>
    </Set>
  </FaultResponse>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</RaiseFault>
```

This response includes:

- The payload containing the error message and an email address for contacting support.
- The HTTP status code returned in the response.
- The reason phrase, which is a short description of the error.

In a fault response, you can also use the Assign Message policy to set the <Payload>, <StatusCode>, and <ReasonPhase> tags.

3.10.11.8 Configuring Default Fault Rules

The DefaultFaultRule acts an exception handler for any error that is not explicitly handled by a FaultRule. If no FaultRules are configured in an API proxy, then the DefaultFaultRule will handle all faults that occur. Default fault handling can be enabled by configuring DefaultFaultRule as a child element of a ProxyEndpoint or a TargetEndpoint.

The TargetEndpoint configuration below is configured with a DefaultFaultRule. The FaultRule indicates that when an error condition is encountered during message processing, a policy called ReturnError executes.

```
<TargetEndpoint name="default">
    <DefaultFaultRule name="fault-rule">
      <Step>
        <Name>ReturnError</Name>
      </Step>
    </DefaultFaultRule>

  <HTTPTargetConnection>
    <URL>http://weather.yahooapis.com</URL>
  </HTTPTargetConnection>
</TargetEndpoint>
```

DefaultFaultRule is typically used to return a generic error message for any unexpected error. For example, you might configure the DefaultFaultRule to return a message that contains support contact information. This

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

serves the dual purpose of providing developer-friendly information while also obfuscating backend URLs or other information that might be used to compromise the system.

For example:

```
<ProxyEndpoint name="MyProxyEndpoint">
  <DefaultFaultRule name="GenericFaultHandler">
  </DefaultFaultRule>
  ....
</ProxyEndpoint>
```

You customize the behavior of the DefaultFaultRule by attaching one or more Policies as processing Steps.

```
<DefaultFaultRule name="GenericFaultHandler">
  <Step><Name>ReturnGenericFaultMessage</Name></Step>
  <AlwaysEnforce>true</AlwaysEnforce>
</DefaultFaultRule>
```

The `AlwaysEnforce` element is provided to enable DefaultFaultRules to execute even when other FaultRules in an API proxy Flow have executed. When the AlwaysEnforce element is set to true, the FaultRule is always enforced, even if fault handling has executed in a previous processing phase.

Recommendation

Set the flag AlwaysEnforce to true on DefaultFaultRule. This enables the DefaultFaultRule to process any faults captured by more specific FaultRules. In this case, you can customize the error response (by adding an HTTP header, for example) on the ProxyEndpoint.

An example of a simple Policy that you might use with a DefaultFaultRule is shown below. The policy is of type AssignMessage, and generates a simple text response for any predefined error condition:

```
<AssignMessage name="ReturnError">
  <Set>
    <Payload type="text/plain">SERVICE UNAVAILABLE. PLEASE CONTACT SUPPORT:
support@company.com.</Payload>
  </Set>
</AssignMessage>
```

Include the `<AlwaysEnforce>` element in the `<DefaultFaultRule>` tag to execute the default fault rule for every error, even if another fault rule has already been executed. The default fault rule is always the last fault rule to execute:

```
<DefaultFaultRule name="fault-rule">
    <Step>
      <Name>ReturnGenericError</Name>
    </Step>
    <AlwaysEnforce>true</AlwaysEnforce>
  </DefaultFaultRule>
```

One use of the default fault rule is to determine the type of error that occurs when you otherwise cannot determine it. For example, your API proxy is failing for an error that you cannot determine. Use the default fault

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

rule to invoke the following AssignMessage policy. This policy writes the `fault.name` value to a header named DefaultFaultHeader in the response:

```
<AssignMessage async="false" continueOnError="false" enabled="true"
name="DefaultFaultRule">
  <DisplayName>DefaultFaultRule</DisplayName>
  <Set>
    <Headers>
      <Header name="DefaultFaultHeader">{fault.name}</Header>
    </Headers>
  </Set>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
  <AssignTo createNew="false" transport="http" type="response"/>
</AssignMessage>
```

You can then view the header in the SAP API Management trace tool or on the response to see what caused the error.

3.10.11.9 Handling policy faults within the current flow

The examples shown so far all use a fault rule on the ProxyEndpoint or TargetEndpoint to handle any policy errors as part of the Error flow. That is because the default value of the `continueOnError` element of a policy is false, meaning that when an error occurs in a policy, control is directed to the Error flow. Once in the Error flow, you cannot return control back to the normal pipeline and you typically return some form of error message to the calling app.

However, if you set the `continueOnError` element to true for a policy, control stays in the current flow and the next policy in the pipeline executes after the policy that caused the error. The advantage to handling the error in the current flow is that you might have a way to recover from the error to complete processing of the request.

Shown below is a VerifyAPIKey policy named verify-api-key with the `continueOnError` element set to true:

```
<VerifyAPIKey async="false" continueOnError="true" enabled="true" name="verify-api-
key">
  <DisplayName>Verify API Key</DisplayName>
  <APIKey ref="request.queryparam.apikey"/>
</VerifyAPIKey>
```

If the API key is missing or invalid, then the VerifyAPIKey policy sets the verifyapikey.verify-api-key.failed variable to true, but processing continues in the current flow.

You then add VerifyAPIKey policy as a step in the PreFlow of the ProxyEndpoint:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<ProxyEndpoint name="default">
  ...
  <PreFlow name="PreFlow">
    <Request>
      <Step>
        <Name>verify-api-key</Name>
      </Step>
      <Step>
        <Name>FaultInFlow</Name>
        <Condition>(verifyapikey.verify-api-key.failed = "true")</Condition>
      </Step>
    </Request>
    <Response/>
  </PreFlow>
</ProxyEndpoint>
```

Notice how the next step in the PreFlow uses a condition to test for the existence of an error. If an error occurred in the VerifAPIKey policy, then the policy named FaultInFlow policy executes. Otherwise, the FaultInFlow policy is skipped. The `FaultInFlow` policy can do many things, such as logging the error, attempting to fix the error, or performing some other action.

3.10.11.10 Triggering an error by using the RaiseFault policy

You can use the RaiseFault policy at any time in a flow to trigger an error. When a RaiseFault policy executes, it terminates the current flow and transfers control to the Error flow.

One use of the RaiseFault policy is to test for a specific condition that another policy might not detect. In the example above, you added a `<Condition>` tag to a PreFlow <Step> tag that caused the policy FaultInFlow to execute if the condition is met. If FaultInFlow is a RaiseFault policy, then control transfers to the Error flow. Or, you might insert a RaiseFault policy in a flow to debug and test your fault rules.

When a RaiseFault policy triggers an error, you can use the following fault rule and condition to process it:

```
<FaultRule name="raisefault_rule">
  <Step>
    <Name>{policy_name}</Name>
  </Step>
  <Condition>(fault.name = "RaiseFault")</Condition>
</FaultRule>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Note that the condition tests for a fault named RaiseFault.

3.10.11.11 Custom Handling of HTTP Error Codes

The configuration shown in the previous section works for application-level errors. To return custom responses for transport-level (that is, HTTP) errors, you must configure a Property on the TargetEndpoint that enables the TargetEndpoint to process HTTP response codes.

By default, HTTP response codes in the 2xx-3xx range are treated as 'success'. HTTP response codes 4xx-5xx are treated as 'failure'. Any response from the backend service with an HTTP response code 400-500 automatically invokes the ErrorFlow. The ErrorFlow automatically returns an error message directly to the requesting client.

To create custom responses to HTTP error codes, you need to override this default behavior. In some cases you might need to return custom errors for HTTP codes 400 and 500. For example, you may need to ensure that all backend service URLs are obfuscated from requesting clients.

To do so, you must first configure the TargetEndpoint to treat HTTP response codes 4xx and 5xx as success codes. By treating those codes as success codes, the TargetEndpoint takes over processing of the response message (rather than pushing the response message into the default ErrorFlow). This enables you to define FaultRules on HTTP error codes that invoke specific policies.

To treat HTTP error codes as success, set the success.codes property on the TargetEndpoint in your API proxy.

```
<TargetEndpoint name="default">
  <HTTPTargetConnection>
    <Properties>
        <Property name="success.codes">4XX, 500</Property>
    </Properties>
    <URL>http://weather.yahooapis.com</URL>
  </HTTPTargetConnection>
</TargetEndpoint>
```

You can then use HTTP response codes 400-499 and 500 to define conditional policy execution that returns a customized response message to the requesting app.

For example, apply the following configuration to a ProxyEndpoint or TargetEndpoint response Flow.

```
<Response>
  <Step>
    <Condition>(response.status.code = 400) or (response.status.code =
500)</Condition>
    <Name>ReturnError</Name>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  </Step>
</Response>
```

The Flow configuration above causes the AssignMessage policy called ReturnError to generate a response message whenever the TargetEndpoint encounters an HTTP response code of 400 or 500.

3.10.11.12 FaultRule Processing

Any number of FaultRules can be defined on ProxyEndpoints and TargetEndpoints. When an error occurs, only the first FaultRule whose condition evaluates to true is enforced.

FaultRules are evaluated in the following order:

Request Path:

3. Fault in ProxyRequest: FaultRules defined at ProxyEndpoint execute
4. Fault in Routing: FaultRules defined at ProxyEndpoint execute
5. Fault in TargetRequest: FaultRules defined at TargetEndpoint execute
6. Fault in outbound request to target URL: FaultRules defined at TargetEndpoint execute

Response Path:

7. Fault in TargetResponse: FaultRules defined at TargetEndpoint execute
8. Fault in ProxyResponse: FaultRules defined at ProxyEndpoint execute
9. Fault in returning response to ProxyEndpoint: FaultRules defined at ProxyEndpoint execute

Policy attachments in a FaultRule are enforced in the order in which the Steps are attached to the FaultRule.

To enable fault handling for an API proxy, fault handlers must be configured to identify and categorize predefined exception conditions, in turn enabling API Platform to execute policies that manage the exception.

3.10.11.13 Fault Taxonomy

API Platform organizes faults into the following categories and subcategories.

Category	Subcategory	Fault Name	Description
Messaging			Failures that occur during the message flow (not including policy failures)
	Custom faults	{fault_name}	Any faults explicitly handled by the API proxy using the RaiseFault policy

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Category	Subcategory	Fault Name	Description
	Response codes	InternalServerError, NotFound	HTTP error codes 5xx, 4xx
	Routing failures	NoRoutesMatched	Failure in selecting a named TargetEndpoint for a request
	Classification failures	NotFound	Failures caused by a request URI that does not match any BasePath for any ProxyEndpoint configurations (that is, no API proxies match the URL in the client app's request)
Transport			HTTP transport-level errors
	Connectivity	ConnectionRefused, ConnectionReset, ConnectionTimeout	Failures occur while establishing network or transport-level connections
	Request validations	ContentLengthMissing, HostHeaderMissing	Faults occur during semantics checks on every request
	Response validations		Faults occur during semantics checks on every response
	IO errors	SSLHandshakeError, ReadTimeout, ReadError, WriteTimeout, WriteError, ChunkError	Read/write errors at client or target endpoints, timeouts, SSL errors, and chunked errors
System			Undefined runtime errors
	Memory	OutOfMemory, GCOverLimit	Memory-related failures
	Thread	RogueTaskTerminated	Failures such as termination of run-away tasks
Step			Faults for each Step (Policy) type are defined in the Policy Reference.

An error is always accompanied by a text description of the reason for the failure. When the system raises a fault, a set of attributes are populated to assist in troubleshooting. A fault includes the following information:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- Reason
- User-defined custom attributes

3.10.12 Pattern matching in conditional statements

This topic explains how to use these pattern matching operators in conditional statements in SAP API Management Edge:

- Matches operator
- JavaRegex operator
- MatchesPath operator

Conditional statements are used in conditional flows, routes, and steps. You can read more about conditional statements in the Conditions reference.

3.10.12.1 Simple matching with the Matches operator

Let's look at the "Matches" or "~" conditional operator first. These two operators are the same -- the English version, "Matches", is considered to be a more readable option.

Summary: The "Matches" operator gives you two possiblities. Either match the string literally, or do a wildcard match with "*". As you might expect, the wildcard matches zero or more characters. Let's see how this works.

The following XML shows a Step condition. It executes the SomePolicy policy when the condition evaluates to true. In this example, we test the variable proxy.pathsuffix, a built-in variable in SAP API Management Edge that stores the path suffix of the request. Note, however, you can test the value of any flow variable that contains a string. So, in this case, if the base path of the incoming request is /animals, and the request is /animals/cat, then the path suffix is the literal string "/cat".

```
<PreFlow name="PreFlow">
    <Request>
        <Step>
            <Condition>(proxy.pathsuffix Matches "/cat")</Condition>
            <Name>SomePolicy</Name>
        </Step>
    </Request>
    <Response/>
</PreFlow>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Question: What proxy path suffix will cause SomePolicy to execute? There's only one possibility.

API call:

```
GET http://artomatic-test.apigee.net/matchtest/cat
```

Does the policy execute? Yes, because the proxy path suffix matches "/cat" exactly. It will not execute if the suffix is /bat or /dog or "/" or anything else.

Now, consider this conditional statement where we use the wildcard character "*":

```
<Condition>(proxy.pathsuffix Matches "/*at")</Condition>
```

API call:

```
GET http://artomatic-test.apigee.net/matchtest/cat
```

Does the policy execute? Yes, because the wildcard matches any character, and "/cat" is a match.

API Call:

```
GET http://artomatic-test.apigee.net/matchtest/bat
```

Does the policy execute? Yes, because the wildcard matches any character, "/bat" is a match.

API Call:

```
GET http://artomatic-test.apigee.net/matchtest/owl
```

Does the policy execute? Certainly not -- although the wildcard matches the "o", the letters "wl" are not matched.

Now, lets move the wildcard to the end of the suffix:

```
<Condition>(proxy.pathsuffix Matches "/cat*")</Condition>
```

API call:

```
GET http://artomatic-test.apigee.net/matchtest/cat
```

Does the policy execute? Yes, because the wildcard matches zero or more of any characters.

API Call:

```
GET http://artomatic-test.apigee.net/matchtest/bat
```

Does the policy execute? No, "/bat" is not a match.

API call:

```
GET http://artomatic-test.apigee.net/matchtest/cat123
```

Does the policy execute? Yes, the wildcard matches zero or more of any characters, therefore "123" produces a match.

API call:

```
GET http://artomatic-test.apigee.net/matchtest/cat/bird/mouse
```

Does the policy execute? Yes, because the wildcard matches zero or more of any characters, so "/bird/mouse" produces a match. Note how an expression like this can get you into trouble because it matches everything after the literal characters!

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.12.1.1 Is the Matches operator case sensitive?

Yes. Assume you have a condition like this:

```
<Condition>(proxy.pathsuffix Matches "/*At")</Condition>
```

API call:

```
GET http://artomatic-test.apigee.net/matchtest/cat
```

Does the policy execute? No, the wildcard matches any letter (regardless of case), but the lowercase "a" does not match "A".

API call:

```
GET http://artomatic-test.apigee.net/matchtest/bAt
```

Does the policy execute? Yes, the case matches.

3.10.12.1.2 How do I escape characters with the Matches operator?

Use the percent "%" character to escape reserved characters. For example:

```
<Condition>(proxy.pathsuffix Matches "/c%*at")</Condition>
```

API call:

```
GET http://artomatic-test.apigee.net/matchtest/cat
```

Does the policy execute? No, the Matches operator is looking for the literal string "c*at".

API call:

```
GET http://artomatic-test.apigee.net/matchtest/c*at
```

Does the policy execute? Yes, this path, although a bit unusual, matches.

3.10.12.2 Finer control with JavaRegex

As you can see, the "Matches" operator is great for simple situations. But you can use another operator, the "JavaRegex" or "~~" operator. These two are the same operator, except JavaRegex is considered to be more readable. It's called JavaRegex because it allows regular expression pattern matching, and SAP API Management Edge follows the same rules as the classes in the java.util.regex package in the Java language. The way the JavaRegex operator works is very different from the Matches operator, so it's important not to confuse the two!

Note:

- If you're used to doing Regex in Perl scripts, it's important to note that the JavaRegex operator in SAP API Management Edge tests the entire subject string. For example, if the subject is "abc" and the regex is "[a-z]" ,

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

there is no match because "[a-z]" matches exactly ONE alpha character. The expression "[a-z]+" would work, as would "[a-z]{3}". This matching differs from the usual application of regular expressions in tools like Perl or grep or Apache's mod_rewrite. JavaRegex uses java.util.regex.Pattern.matches(str), which is defined to return positive if the entire string (str) matches the pattern. The regex in Perl or mod_rewrite are applied to "any" part of the string. They return true if the pattern matches anywhere. See the example Matching arbitrary strings with JavaRegex.

- This section is not intended to be a detailed introduction to regular expressions. The main intent is to illustrate a few simple examples, and highlight the difference between the Matches and JavaRegex operators. If you'd like to know more about regular expressions, many resources exist for you to explore.

Summary: The "JavaRegex" operator lets you use regular expression syntax in conditional statements.

The following code shows a Step condition. It executes the SomePolicy policy if the condition evalutes to true. In this example, we test the variable proxy.pathsuffix, a built-in variable in SAP API Management Edge that stores the path suffix of the request. Note, however, you can test the value of any flow variable that contains a string. So, in this case, if the base path of the incoming request is /animals, and the request is /animals/cat, then the path suffix is the literal string "/cat".

```
<PreFlow name="PreFlow">
    <Request>
        <Step>
            <Condition>(proxy.pathsuffix JavaRegex "/cat")</Condition>
            <Name>SomePolicy</Name>
        </Step>
    </Request>
    <Response/>
</PreFlow>
```

Question: What proxy path suffix will cause SomePolicy to execute? Just like with the Matches operator, there's only one possibility in this case.

API call:

`GET http://artomatic-test.apigee.net/matchtest/cat`

Does the policy execute? Yes, because the proxy path suffix matches "/cat" exactly. It will not execute if the suffix is /bat or /dog or anything else.

Now, let's create a regular expression using "*" quantifier. This quantifier matches zero or more of the preceeding character.

```
<Condition>(proxy.pathsuffix JavaRegex "/c*t")</Condition>
```

API call:

`GET http://artomatic-test.apigee.net/matchtest/cat`

Does the policy execute? No! The "*" quantifier matches zero or more of the preceeding character, which is a "c".

API Call:

`GET http://artomatic-test.apigee.net/matchtest/ccccct`

Does the policy execute? Yes, because the wildcard matches zero or more of the preceeding character.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Next, we use the "?" quantifier, which matches the preceeding character once, or not at all.

```
<Condition>(proxy.pathsuffix JavaRegex "/ca?t")</Condition>
```

API call:

```
GET http://artomatic-test.apigee.net/matchtest/cat
```

Does the policy execute? Yes. The "?" quantifier matches zero or one occurance of the preceeding character, which is an "a".

API Call:

```
GET http://artomatic-test.apigee.net/matchtest/ct
```

Does the policy execute? Yes. The "?" quantifier matches one or none of the preceeding character. In this case, there is no "a" character, so the condition evaluates to true.

API Call:

```
GET http://artomatic-test.apigee.net/matchtest/caat
```

Does the policy execute? No. The "?" quantifier matches one of the preceeding character, which is an "a".

Next, we use the "[abc]" or "grouping" style of regex expression. It matches the characters "a" or "b" or "c".

```
<Condition>(proxy.pathsuffix JavaRegex "/[cbr]at)</Condition>
```

API call:

```
GET http://artomatic-test.apigee.net/matchtest/cat
```

Does the policy execute? Yes. We're using regular expressions here, and the "[cbr]" expression matches a "c", "b", OR "r". These calls are also matches:

```
GET http://artomatic-test.apigee.net/matchtest/bat
```

```
GET http://artomatic-test.apigee.net/matchtest/rat
```

But this is not a match:

```
GET http://artomatic-test.apigee.net/matchtest/mat
```

3.10.12.2.1 Is the JavaRegex operator case sensitive?

Yes. Assume you have a condition like this:

```
<Condition>(proxy.pathsuffix Matches "/ca?t")</Condition>
```

API call:

```
GET http://artomatic-test.apigee.net/matchtest/cat
```

Does the policy execute? Yes, the regex matches zero or one of the preceeding character, which is "a".

API call:

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
GET http://artomatic-test.apigee.net/matchtest/cAt
```

Does the policy execute? No, because the capital "A" does not match lowercase "a".

3.10.12.3 Matching path fragments with the MatchesPath operator

The MatchesPath operator can also be specified like this "~/". It looks a little bit like the Matches (~) and the JavaRegex (~~) operators. But MatchesPath is entirely different.

Just remember that this operator looks at a path as a series of parts. Therefore, if the path is: /animals/cats/wild, you can think of the path as consisting of the parts "/animals", "/cats", and "/wild".

The MatchesPath operator lets you use two wildcard notations: a single asterisk (*) and a double asterisk (**). The single asterisk matches one path element. The double asterisk matches one or many path elements.

Let's look at an example. In this example, we test the variable proxy.pathsuffix, a built-in variable in SAP API Management Edge that stores the path suffix of the request. Note, however, you can test the value of any flow variable that contains a string.

```
<PreFlow name="PreFlow">
    <Request>
        <Step>
            <Condition>(proxy.pathsuffix MatchesPath "/animals/*)</Condition>
            <Name>SomePolicy</Name>
        </Step>
    </Request>
    <Response/>
</PreFlow>
```

Question: What proxy path suffix will cause SomePolicy to execute?

API call:

```
GET http://artomatic-test.apigee.net/matchtest/animals
```

Does the policy execute? No, because condition requires another path element after "/animals", as specified by "/*".

API call:

```
GET http://artomatic-test.apigee.net/matchtest/animals/
```

Does the policy execute? Yes, the path does have another path element (the part after "/animals/"), but it's just empty.

API Call:

```
GET http://artomatic-test.apigee.net/matchtest/animals/cats
```

Does the policy execute? Yes, because the path clearly has an element ("/cats") that comes after "/animals"

API Call:

```
GET http://artomatic-test.apigee.net/matchtest/animals/cats/wild
```

Does the policy execute? No, because the single asterisk only matches one path element, and this API has more than one element after "/animals".

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Now let's use the double asterisk:

```
<PreFlow name="PreFlow">
    <Request>
        <Step>
            <Condition>(proxy.pathsuffix MatchesPath "/animals/**)</Condition>
            <Name>SomePolicy</Name>
        </Step>
    </Request>
    <Response/>
</PreFlow>
```

Question: What proxy path suffix will cause SomePolicy to execute?

API call:

`GET http://artomatic-test.apigee.net/matchtest/animals`

Does the policy execute? No, because condition requires at least one following path element specified by "/**".

API call:

`GET http://artomatic-test.apigee.net/matchtest/animals/`

Does the policy execute? Yes, the path does have another path element (the part after "/animals/"), but it's just empty.

API Call:

`GET http://artomatic-test.apigee.net/matchtest/animals/cats`

Does the policy execute? Yes, because the path has at least one element that comes after "/animals"

API Call:

`GET http://artomatic-test.apigee.net/matchtest/animals/cats/wild`

Does the policy execute? Yes, because the path has more than one element that comes after "/animals"

3.10.12.3.1 Mixing asterisks

You can use combinations of the single (*) and double (**) asterisk to further refine your path matching.

```
<PreFlow name="PreFlow">
    <Request>
        <Step>
            <Condition>(proxy.pathsuffix MatchesPath
"/animals/*/wild/**)</Condition>
            <Name>SomePolicy</Name>
        </Step>
    </Request>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        <Response/>
    </PreFlow>
```

API call:

All of these API calls will produce a match:

```
GET http://artomatic-test.apigee.net/matchtest/animals/cats/wild/
```

and

```
GET http://artomatic-test.apigee.net/matchtest/animals/dogs/wild/austrailian
```

and

```
GET http://artomatic-test.apigee.net/matchtest/animals/birds/wild/american/finches
```

3.10.12.4 Examples

3.10.12.4.1 A practical example: Ignore "/" at the end of a path

SAP API Management Edge developers commonly want to handle both of these path suffixes: "/cat" and "/cat/". This is because some users or clients might call your API with the extra slash on the end of the path, and you need to be able to handle that in your conditional statements.

If you prefer, you can achieve this without using Regex like this:

```
    <PreFlow name="PreFlow">
        <Request>
            <Step>
                <Condition>((proxy.pathsuffix = "/cat") OR (proxy.pathsuffix =
"/cat/")</Condition>
                <Name>SomePolicy</Name>
            </Step>
        </Request>
        <Response/>
    </PreFlow>
```

This is a good option. It is clear and readable.

You can do the same thing with Regex, however, like this. The parentheses are used to group the regex part of the statement, but they are not required.

```
<Condition>(proxy.pathsuffix JavaRegex "/cat(/?)"</Condition>
```

API Calls:

```
GET http://artomatic-test.apigee.net/matchtest/cat
```

or

```
GET http://artomatic-test.apigee.net/matchtest/cat/
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Does the policy execute? Yes. Note that in a regular expression, the "?" character means: match zero or one of the preceeding character. Therefore, both "/cat" and "/cat/" are matches.

API Call:

```
GET http://artomatic-test.apigee.net/matchtest/cat/spotted
```

Does the policy execute? No. The regular expression matches zero or only one occurance of the preceeding character, and nothing else is allowed.

3.10.12.4.2 Matching arbitrary strings with JavaRegex

In all of the examples in this topic, we show how to match one of the built-in flow variables: proxy.pathsuffix. It's good to know that you can do pattern matching on any arbitrary string or flow variable, whether or not it's a built-in flow variable like proxy.pathsuffix.

If for example you have a condition that tests an arbitrary string, perhaps a string returned in a backend payload, or a string returned from an authentication server lookup, you can use matching operators to test it. If you use JavaRegex, the regular expression will be compared against the entire subject string. If the subject is "abc" and the regular expression is "[a-z]", there is no match, because "[a-z]" matches exactly one alpha character. The expression "[a-z]+" works, as does "[a-z]*", and "[a-z]{3}.

Let's look at a concrete example. Suppose the authentication server returns a list of roles as a comma-delimted string: "editor, author, guest".

To test the presence of the editor role, this construction will not work, because "editor" is only part of the entire string.

```
<Condition>returned_roles ~~ "editor"</Condition>
```

However, this construction will work:

```
<Condition>returned_roles ~~ ".*\beditor\b.*")</Condition>
```

It works because it takes into account word breaks and any other parts of the string with the .* prefix and suffix.

In this example, you can also test for "editor" with the Matches operator:

```
<Condition>returned_roles ~~ "*editor*")</Condition>
```

However, in cases where you need more precision, JavaRegex is often a better choice.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.13 Resource Files

Many policy types defined by SAP API Management rely on resources. Resources are the files that implement the code or configuration to be executed by a policy when attached to an API proxy. In some cases, as with JavaScript and JavaCallout, a policy simply defines an attachment point in an API proxy where some code should execute. A JavaScript or JavaCallout policy is a pointer to a resource.

> **Note**
>
> In the management UI, resources scoped to an API proxy are listed in the Scripts section of the Navigation pane.

These resources can be stored in any of three locations in an organization.

- API proxy: When stored in an API proxy, resources are available only to that API proxy.
- Environment: When stored in an environment (for example, test or prod), the resource is available to any API proxy deployed in the same environment.
- Organization: When stored in an organization, the resource is available to any API proxy deployed in any environment.

In addition to storing resources at the proxy level, API Services also provides repositories for these resources at the organization and the environment level. By storing resources at the organization or the environment level, you enable reuse and sharing of code and configuration across the API proxies deployed in your organization and across different environments.

> **Note**
>
> The example API calls in this topic use cURL. You can also use the SmartDocs management API reference topic SmartDocs to make the same calls.

3.10.13.1 Resource types

Resource repositories support the following resource types:

- `jsc`: compiled JavaScript, referenced by policies of type Javascript
- `java`: Java classes, referenced by policies of type JavaCallout
- `py`: Python scripts, referenced by policies of type Python
- `node`: Node.js files, including the main Node.js file, related source files, and module dependencies
- `wsdl`: WSDL files, referenced by policies of type MessageValidation
- `xsd`: XML schemas, referenced by policies of type MessageValidation
- `xsl`: XSLT transformations, referenced by policies of type XSL

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The repositories are available at the following URIs, as described in the Resource files API:

```
/organizations/{org_name}/resourcefiles
/organizations/{org_name}/environments/{environment_name}/resourcefiles
/organizations/{org_name}/apis/{api_name}/revisions/{revision_number}/resources
```

> **Note**
>
> You can also add resources at the API proxy scope in the SAP API Management UI, in the proxy editor Develop view.

For example, all JavaScript files available across an organization are stored under:

```
/organizations/{org_name}/resourcefiles/jsc
```

JavaScript stored in the /organizations collection is available to any API proxy running in any environment.

You can list all available resources by calling GET on the collection:

```
$ curl https://<host:port>/v1/organizations/{org_name}/resourcefiles/{resource_type}
```

The following request lists all JavaScript resources at the organization level:

```
$ curl https://<host:port>/v1/organizations/myorg/resourcefiles/jsc
```

The following request lists all JavaScript resources at the environment level, in the environment called prod:

```
$ curl https://<host:port>/v1/organizations/myorg/environments/prod/resourcefiles/jsc
```

The following request lists all JavaScript resources in an API proxy revision (the most specific level):

```
$ curl
https://<host:port>/v1/organizations/myorg/apis/weatherapi/revisions/6/resourcefiles/j
sc
```

Each request returns a list of resource names.

Sample response:

```
<List>
  <Item>JavaScriptMashUp</Item>
  <Item>JavaScriptClient</Item>
</List>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.13.2 Populating resource repositories

> **Note**
>
> When updating an existing resource in a deployed API proxy, you must redeploy the API proxy after to ensure the resource changes are picked up. Newly added resources do not require API proxy redeployment.

Let's assume that you have a simple snippet of JavaScript that you need to make available to API proxies across your organization. The following sample JavaScript sets the HTTP request path to the value of the proxy.basepath variable. It's just one example of a resource that you might need to put in the repository.

```
request.setHeader("RequestPath", flow.getVariable("proxy.basepath"));
```

To create a resource, you call the POST method, submitting the body of the resource file, and identifying the resource's name and type as query parameters.

To make the JavaScript available to API proxies running in any environment in your organization:

```
$ curl -X POST -H "Content-type:application/octet-stream" -d \
'request.setHeader("RequestPath", flow.getVariable("proxy.basepath"));' \
https://<host:port>/v1/organizations/myorg/resourcefiles?"name=pathSetter&type=jsc" \
-u email:password
```

Sample response:

```
{
  "name" : "pathSetter",
  "type" : "jsc"
}
```

You can also upload resources as files from your local machine as follows. It's important to use -F for the binary upload in cURL in order for the environment- and organization-scoped JavaScript files to be accessible by the JavaScript policy.

```
$ curl -X POST -H "Content-type:multipart/form-data" -F file=@pathSetter.js \
https://<host:port>/v1/organizations/myorg/resourcefiles?"name=pathSetter&type=jsc" \
-u email:password
```

The JavaScript resource named pathSetter is now available to be referenced by policies of type JavaScript running in any environment in the organization.

For example, to attach the JavaScript to the Request PostFlow, create a policy called PathSetterPolicy.xml that references the filepathSetter.js:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<Javascript name='PathSetterPolicy' timeLimit='200'>
    <ResourceURL>jsc://pathSetter.js</ResourceURL>
</Javascript>
```

Then reference the policy in the Endpoint configuration:

```
<PostFlow>
  <Request>
    <Step><Name>PathSetterPolicy</Name></Step>
  </Request>
<PostFlow>
```

3.10.13.3 Adding Java resources

You can add compiled Java resources as JAR files using multiple options in cURL, such as -T, --data-binary, or -F option (not the -d option). For example:

```
curl -v -u {user}:{password} -H "Content-Type: application/octet-stream" \
-X POST --data-binary @{jar_file} \
"http://{mgmt_server}:{port}/v1/organizations/{org}/resourcefiles?name={jar_file}&type=java"
```

```
curl -v -u {user}:{password} -H "Content-Type: application/octet-stream" \
-X POST -T "{jar_file}" \
"http://{mgmt_server}:{port}/v1/organizations/{org}/resourcefiles?name={jar_file}&type=java"
```

```
curl -v -u {user}:{password} -H "Content-Type: multipart/form-data" \
-X POST -F "file=@{jar_file}" \
"http://{mgmt_server}:{port}/v1/organizations/{org}/resourcefiles?name={jar_file}&type=java"
```

3.10.13.4 Resource name resolution

SAP API Management resolves resource names from the most specific to the most general scope. Resource names are resolved "up the chain", from the API proxy level, to the environment level, to the organization level.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Let's say that you have populated the same resource in two different repositories — the organization and the environment:

```
$ curl -X POST -H "Content-type:application/octet-stream" -d \
'request.setHeader("RequestPath", flow.getVariable("proxy.basepath"));' \
https://<host:port>/v1/organizations/myorg/resourcefiles?"name=pathSetter&type=jsc" \
-u email:password

$ curl -X POST -H "Content-type:application/octet-stream" -d \
'request.setHeader("RequestPath", flow.getVariable("proxy.basepath"));' \
https://<host:port>/v1/organizations/myorg/environments/prod/resourcefiles?"name=pathSetter&type=jsc" \
-u email:password
```

Let's take a scenario where the same resource is available in two repositories, at the organization level and at the environment level:

```
https://<host:port>/v1/organizations/myorg/resourcefiles/jsc/pathSetter
https://<host:port>/v1/organizations/myorg/environments/prod/resourcefiles/jsc/pathSetter
```

Now imagine that an API proxy is configured with the following Policy:

```
<Javascript name='PathSetterPolicy' timeLimit='200'>
    <ResourceURL>jsc://pathSetter.js</ResourceURL>
</Javascript>
```

> **Hint**
>
> The policy reference cannot explicitly resolve to a repository. The first resource whose name matches the resource name in the policy is resolved.

So, when the API proxy is deployed in the environment prod, the Policy will resolve the pathSetter.js resource:

```
/organizations/myorg/environment/prod/resourcefiles/jsc/pathSetter
```

When deployed in the environment test, the Policy will resolve:

```
/organizations/myorg/resourcefiles/jsc/pathSetter
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.13.5 Retrieving resources

You can view a resource body by calling the GET method on it:

```
$ curl -X GET -H "Accept:application/json"
https://<host:port>/v1/organizations/myorg/resourcefiles/jsc/pathSetter" \
-u email:password
```

Sample response:

```
request.setHeader("RequestPath", flow.getVariable("proxy.basepath"));
```

3.10.14 JMS Integration

3.10.14.1 Overview

If you use Java Message Service (JMS) with Java Naming and Directory Interface (JNDI), you can make RESTful calls from JMS to backend services through SAP API Management.

This document describes how to configure SAP API Management to work with a JMS host and how to create an API proxy that handles incoming JMS messages.

Regardless of the JMS provider you use—for example, whether you're using Apache ActiveMQ or JBoss HornetQ (or neither)—the SAP API Management supports only Java Naming and Directory Interface (JNDI) to connect to the JMS provider. If a JMS provider doesn't support JNDI, it can't be used with SAP API Management.

PUT operations to a JMS queue

SAP API Management's default JMS integration supports receiving messages from a JMS queue. If you want to do PUT operations to a JMS queue, use a Java Callout policy.

3.10.14.2 Use Case

You can send messages from JMS queues to an SAP API Management API resource. For example, you might need an API proxy endpoint to handle incoming JMS messages from multiple JMS queues, apply logic to convert JMS messages to HTTP, forward the request to a target endpoint, and return the response as JMS.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.14.3 Creating a JMS Host

To use JMS with SAP API Management, you configure the JMS connection details by creating a JMS host . The JMS host then can be referenced from JMS API proxy endpoints by using the <JMSHost> or <JMSResponseHost> tags. All JMS API proxies share the same JMS host.

```
curl –H “Content-Type: application/xml” –u : –X POST
“http://{host}:8080/v1/organizations/environments/jmshosts" –d
“<JMSHost name="jmshost1">
<Description></Description>
<ConnectionFactoryClass>org.apache.activemq.jndi.Active
</ConnectionFactoryClass>
<ConnectionFactory>FooFactory</ConnectionFactory>
<ConnectionURL>tcp://(IP Address):61616</ConnectionURL>
<Context>JmsContext</Context>
<ContextUsername>abc</ContextUsername>
<ContextPassword>password</ContextPassword>
<ConnectionUsername>admin</ConnectionUsername>
<ConnectionPassword>password</ConnectionPassword>
<Connections>10</Connections>
</JMSHost>”
```

3.10.14.4 Request payload elements

Name	Description
name	Name of the JMS host.
Description	Description of the JMS host.
ConnectionFactoryClass	Provider specific implementation of JMS Connection factory class.
ConnectionFactory	Property used to select the registry service provider as the initial context and specifies the class name of the initial context factory for the provider. Note that it is not used by the provider.
ConnectionURL	URL of the JNDI provider.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Context	Initial JNDI context and values are specific to JMS provider.
ContextUsername	User name to access the JNDI context.
ContextPassword	Password to access the JNDI context.
ConnectionUsername	User name to access the JMS provider.
ConnectionPassword	Password to access the JMS provider.
Connection	Number of physical TCP connections toward JMS provider.

You might need to remove a JMS host associated with an environment. To do this, execute the following API call:

```
curl –u : “http://{host}:8080/v1/organizations/environments/jmshosts/{jmshost_name}” -X DELETE
```

3.10.14.5 Creating a JMS proxy endpoint

The ProxyEndpoint and JmsProxyConnection elements define a set of JMS transport properties. These properties can be used to set network configurations to connect to the JMS provider. The SAP API Management JMS API proxy endpoint supports only receiving messages from a JMS queue. Other JMS operations are not supported.

Properties are set on a JMS API proxy as follows:

```
<ProxyEndpoint name=”default” proxyType="jmsConnection">
<!-- "proxyType” is mandatory if you want to create JMS endpoints -->
  <JmsProxyConnection state=”on”>
    <JMSHost>defaultHost</JMSHost> <!-- this is the JMS host created in prev section -->
    <Destination type="queue">dynamicQueues/myqueue</Destination>
    <MessageSelector></MessageSelector>
    <DefaultResponseQueue>dynamicQueues/defaultResponseQueue</DefaultResponseQueue>
    <JMSResponseHost>defaultResponseHost</JMSResponseHost>
  </JmsProxyConnection>
</ProxyEndpoint>
```

Ensure that you define the attribute `proxyType="jmsConnection"` in order to create JMS proxy endpoints. Also, ensure that you have the JMS provider-specific client library in the classpath.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.14.6 Property description

The following table provides the description of all JmsProxyConnection properties that are available for JMS implementation.

Property name	Description	Required ?
JmsProxyConnection	JMS network details associated with the API proxy. You can dynamically set to subscribe/unsubscribe to a given queue by defining an attribute state="on/off". By default it is "on".	Yes
JMSHost	JMS host defined for the JMS proxy.	Yes
Destination	Queue path to which the subscribe listens to the messages.	Yes
MessageSelector	Selector pattern on queues based on JMS headers.	No
DefaultResponseQueue	Name of the JMS queue where response from target is published.	No
JMSResponseHost	JMS response host defined for the JMS proxy. If specified, all responses are sent to the queues over this host.	No

Note

All HTTP headers matching standard JMS headers will be set "as is" and other HTTP headers will be set as JMS properties in the response message used by the JMS proxy.

3.10.14.7 Setting JMS headers

You can use the AssignMessage policy to specify the following headers in a JMS proxy.

X-SAP-Ignore-JMSResponse - By default, the API proxy puts the response on the JMS response queue specified in the JMSReplyTo header. However, if you want the backend service to handle putting the response on the queue specified in JMSReplyTo—or something else outside of the API proxy such as a Java callout or another API proxy—add the X-SAP-Ignore-JMSResponse header and set it to true:

```
<Header name="X-SAP-Ignore-JMSResponse">true</Header>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Where to set the header: If the proxy is calling a backend service, the header must be set after the backend service is called. If there is no backend service, such as when you use a Java callout or another API proxy to put the message on the response queue, the header can be set anywhere in the flow of the API proxy (the API proxy you don't want putting the response on the queue).
For more information about API proxy flows, see Configuring flows.

For messages placed in the ResponseQueue:
`JMSExpiration` - Time in milliseconds after which the message expires.
`JMSDeliveryMode` - A value of 1 (non-persistent) or 2 (persistent).

3.10.15 Adding CORS Support to an API proxy

CORS (Cross-origin resource sharing) is a standard mechanism that allows JavaScript XMLHttpRequest (XHR) calls executed in a web page to interact with resources from non-origin domains. CORS is a commonly implemented solution to the "same-origin policy" that is enforced by all browsers. For example, if you make an XHR call to the Twitter API from JavaScript code executing in your browser, the call will fail. This is because the domain serving the page to your browser is not the same as the domain serving the Twitter API. CORS provides a solution to this problem by allowing servers to "opt-in" if they wish to provide cross-origin resource sharing.

Recommendation

Most modern browsers support CORS.For an in-depth description of CORS; see the Cross-Origin Resource Sharing W3C Recommendation.

3.10.15.1 Typical Use Case for CORS

The following JQuery code calls the Yahoo Weather API. If executed from within the context of a browser (a web page), the call will fail because of the same-origin policy:

```
<script>
var url = "http://weather.yahooapis.com/forecastrss?w=12797282";
$(document).ready(function(){
  $("button").click(function(){
    $.ajax({
        type:"GET",
        url:url,
        async:true,
        dataType: "xml",
           success: function(xml) {
              // Parse the response.
              // Do other things.
           },
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
            error: function(xhr, status, err) {
               // This is where we end up!
             }
      });
    });
  });
  </script>
```

One solution to this problem is to create an SAP API Management API Platform proxy that calls the Yahoo API on the back end. Remember that API Platform sits between the client (a browser in this case) and the backend API (Yahoo Weather). Because the API proxy executes on the server, not in a browser, it is able to call Yahoo Weather successfully. Then, all you need to do is attach CORS headers to the TargetEndpoint response. As long as the browser supports CORS, these headers signal to the browser that it's okay to "relax" its same-origin policy, allowing the cross-origin API call to succeed.

Once the proxy with CORS support is created, you can call the API proxy URL instead of the backend service in your client-side code. For example:

```
<script>
var url = "http://<host:port>/v1/my-weather-api/forecastrss?w=12797282";
$(document).ready(function(){
  $("button").click(function(){
    $.ajax({
        type:"GET",
        url:url,
        async:true,
        dataType: "xml",
           success: function(xml) {
              // Parse the response.
              // Do other things.
           },
           error: function(xhr, status, err) {
              // This time, we do not end up here!
            }
    });
  });
});
</script>
```

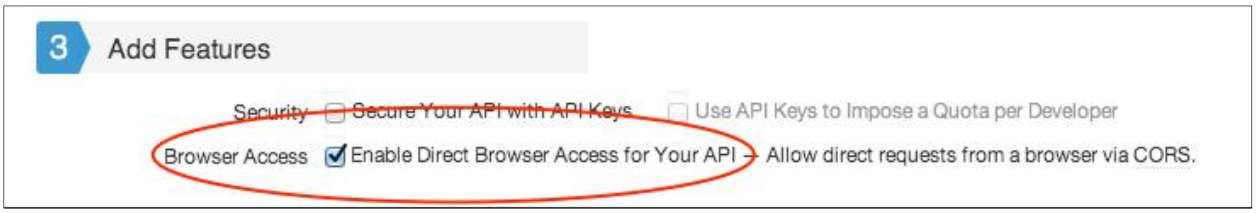
Recommendation

In the simplest case, you can return CORS headers to the client and the cross-origin request will work. More complicated cases exist where a "preflight" request is required. You can read about preflight CORS requests in Cross-Origin Resource Sharing W3C Recommendation, as well as in numerous articles and blogs. See also "Handling CORS preflight requests" below

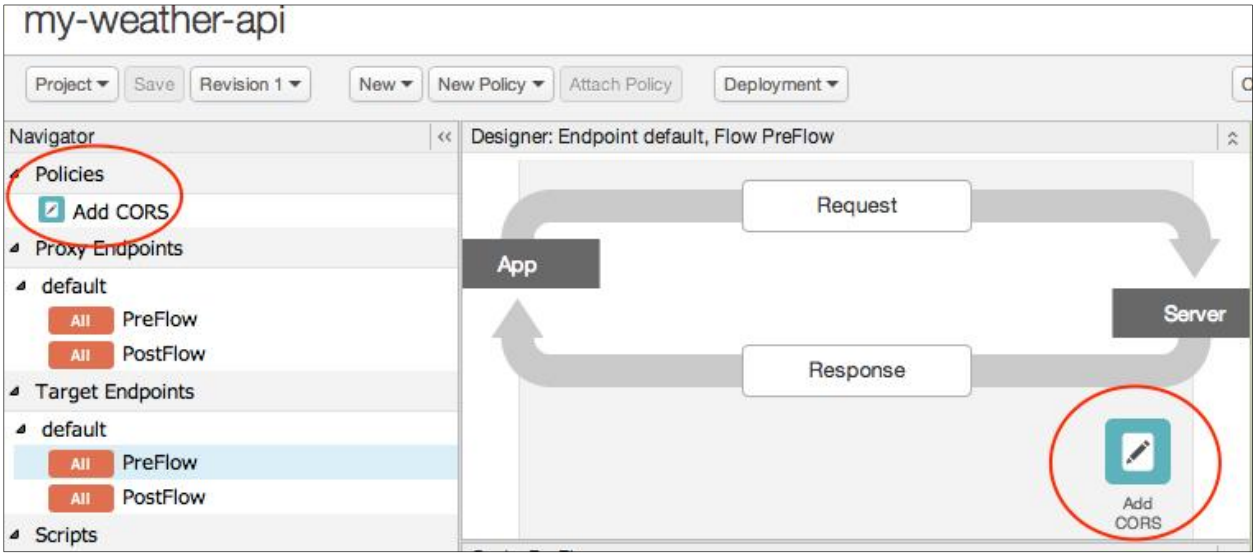
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.15.2 Attaching an Add CORS Policy to a new API Proxy

You can add CORS support to an API proxy by attaching an "Add CORS" policy to the API proxy when you create it. To add this policy, select the Browser Access checkbox in the Add API Proxy dialog, as shown in the following figure.



When you select this checkbox, a policy called Add CORS is automatically added to the system and attached to the TargetEndpoint response preflow, as shown in the following figure:



The Add CORS policy is implemented as an AssignMessage policy, which adds the appropropriate headers to the response. Here is the XML for the Add CORS policy:

```
<AssignMessage async="false" continueOnError="false" enabled="true" name="Add-CORS">
    <DisplayName>Add CORS</DisplayName>
    <FaultRules/>
    <Properties/>
    <Add>
        <Headers>
            <Header name="Access-Control-Allow-Origin">*</Header>
            <Header name="Access-Control-Allow-Headers">origin, x-requested-with, accept</Header>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
            <Header name="Access-Control-Max-Age">3628800</Header>
            <Header name="Access-Control-Allow-Methods">GET, PUT, POST,
DELETE</Header>
        </Headers>
    </Add>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
    <AssignTo createNew="false" transport="http" type="response"/>
</AssignMessage>
```

Basically, the headers let the browser know which origins it will share its resources with (in this case "all origins"), which methods it accepts, and so on. You can read more about these CORS headers in the Cross-Origin Resource Sharing W3C Recommendation.

3.10.15.3 Adding CORS Headers to an Existing Proxy

You need to manually create a new Assign Message policy and copy the code for the Add CORS policy listed in the previous section into it. Then, attach the policy to the response preflow of the TargetEndpoint of the API proxy. You can modify the header values as needed. For more information on creating and attaching policies, see Policy Attachment and Enforcement.

3.10.15.4 Handling CORS Preflight Requests

CORS preflight refers to sending a request to a server to verify if it supports CORS. Typical preflight responses include which origins the server will accept CORS requests from, a list of HTTP methods that are supported for CORS requests, headers that can be used as part of the resource request, the maximum time preflight response will be cached, and others. If the service does not indicate CORS support or does not wish to accept cross-origin requests from the client's origin, the cross-origin policy of the browser will be enforced and any cross-domain requests made from the client to interact with resources hosted on that server will fail.

Typically, CORS preflight requests are made with the HTTP OPTIONS method. When a server that supports CORS receives an OPTIONS request, it returns a set of CORS headers to the client to indicate its level CORS support. As a result of this handshake, the client knows what it is allowed to request from the non-origin domain.

For more information on preflight, refer to the Cross-Origin Resource Sharing W3C Recommendation. There are in addition numerous blogs and articles on CORS that you can refer to.

SAP API Management does not include a CORS preflight solution out of the box, but it is possible to implement, as described in this section. The objective is for the proxy to evaluate an OPTIONS request in a conditional flow. The proxy can then send an appropriate response back to the client.

Let's look at a sample flow, and then discuss the parts that handle the preflight request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ProxyEndpoint name="default">
    <Description/>
    <Flows>
        <Flow name="OptionsPreFlight">
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
            <Request/>
            <Response>
                <Step>
                    <Name>Add-CORS</Name>
                </Step>
            </Response>
        <Condition>request.verb == "OPTIONS"</Condition>
        </Flow>
    </Flows>

    <PreFlow name="PreFlow">
        <Request/>
        <Response/>

    </PreFlow>
    <HTTPProxyConnection>
        <BasePath>/v1/cnc</BasePath>
        <VirtualHost>default</VirtualHost>
        <VirtualHost>secure</VirtualHost>
    </HTTPProxyConnection>
    <RouteRule name="NoRoute">
        <Condition>request.verb == "OPTIONS"</Condition>
    </RouteRule>
    <RouteRule name="default">
        <TargetEndpoint>default</TargetEndpoint>
   </RouteRule>
   <PostFlow name="PostFlow">
        <Request/>
        <Response/>
    </PostFlow>
</ProxyEndpoint>
```

The key parts of this ProxyEndpoint are as follows:

- A RouteRule is created to a NULL target with a condition for the OPTIONS request. Note that there is no TargetEndpoint specified. If the OPTIONS request is received, the proxy immediately returns the CORS headers in a response to the client (bypassing the actual default "backend" target). For details on flow conditions and RouteRule, see Flow variables and conditions.

  ```
  <RouteRule name="NoRoute">
      <Condition>request.verb == "OPTIONS"</Condition>
  </RouteRule>
  ```

- An OptionsPreFlight flow is created that adds an Add CORS policy, containing the CORS headers, to the flow if an OPTIONS request is received.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<Flow name="OptionsPreFlight">
            <Request/>
            <Response>
                <Step>
                    <Name>Add-CORS</Name>
                </Step>
            </Response>
        <Condition>request.verb == "OPTIONS"</Condition>
 </Flow>
```

Recommendation

RouteRules are evaluated in the order specified in the ProxyEnpoint configuration. You should always have the default (no condition) Route at the end. Otherwise, if at the top, it will always match and never evaluate the other Route possibilities.

3.10.16 Exposing a SOAP Service as an API Proxy

This topic explains how to create API proxies for SOAP-based web services. You can create two kinds of SOAP proxies in SAP API Management Edge. One generates a RESTful interface to the backend SOAP service and the other performs a "pass through" of the SOAP message to the backend. Both techniques are described in this topic.

3.10.16.1 Creating a RESTful API Proxy to a SOAP-Based Service

This section explains how to create a RESTful SOAP API proxy with the REST to SOAP to REST option in the Create New Proxy dialog.

3.10.16.1.1 Overview

The REST to SOAP to REST option processes the WSDL to generate a RESTful API proxy. SAP API Management Edge determines from the WSDL the service's supported operations, input parameters, and so on. SAP API Management Edge "guesses" which HTTP method to use for each operation. Typically, SAP API Management Edge translates operations into GET requests, which have the advantage of being cacheable. SAP API Management Edge also sets up the backend target endpoint, which can vary per SOAP operation.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.16.1.2 Basic Steps

1. From the Dashboard, choose + API Proxy.
2. In the Build a Proxy wizard, select SOAP service.
3. Click Next.
4. In the Details page, make these selections. You must click Validate after selecting a WSDL.

In this field	do this
WSDL	Select the source of the WSDL. · URL - Enter the URL of the WSDL you wish to use. · File - Choose a WSDL file on your file system. In cases where there are additional dependent files, you can select all of them. · Example URL - Select from a list of WSDLs for publicly available web services. These are handy for trying out the SOAP / API proxy features of SAP API Management Edge.
Proxy Name	This is name for the proxy you're creating.
Proxy Base Path	The Proxy Base Path is a URI fragment that uniquely identifies the API that is exposed by this API proxy. API Services uses the Base Path URI to match and route incoming request messages to the proper API proxy. (The Base Path is appended to the domain of the API, which is automatically generated based on your organization name and the environment where the API proxy is deployed.) It's a best practice to include a version number in the project name, for example, `/v1/weather`. This will determine how your API is invoked by consumer apps. Note: The Proxy Base Path defaults to the value specified for Proxy Name converted to all lower case unless you explicitly edit the content in the Proxy Base Path field.
Description	A brief description of the proxy.

5. Click Next.
6. In the WSDL page, select the API proxy type REST to SOAP to REST.

> **Note**
>
> A table appears listing the operations that SAP API Management "discovered" in the WSDL file. You can select and configure which operations you wish to incorporate into your API proxy.

> **Recommendation**
>
> Operations on the callback porttype cannot be converted to REST resources because API Platform does not support asynchronous operations for this purpose.

7. Select from the Port Type column which set of operations you wish to use. In WSDL, port type elements define the operations that you can call on a web service.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8. Optionally change the HTTP Method associated with the operation.

   Note

   SAP API Management makes a "best guess" in determining the HTTP method to use for each operation. GET is generally preferred because GET requests can be cached.

9. Optionally change the REST API path for an operation. The path will be used as the resource name in the API proxy URL.

   Note

   Any parameters for an operation are listed in the REST API Parameters column. For example, the weather API shown in the previous figure includes a ZIP parameter for certain operations. In the API proxy, you specify that value as a query parameter.

10. Click through the rest of the wizard to add security, select virtual hosts, and deployment environment.
11. In the Build page, click Build and Deploy. SAP API Management Edge generates and deploys the new API proxy based on the WSDL.
12. Go to the summary page for the new API proxy. Note that a set of resources have been constructed based on the operations discovered in the WSDL file.

    Recommendation

    On the proxy's Overview page, the Resources list provides a detailed description of the new API, its operations, and parameters. You can think of this representation as the API's reference documentation. SAP API Management generates this view of the API model automatically for you. Simply expand a resource to see its description and information about its input parameters, data types. Also listed are any enumeration facets that apply to an input parameter. Enumeration facets specify the list of the acceptable values for an input parameter.

3.10.16.1.3 About the Final Proxy

When SAP API Management generates an API proxy based on a WSDL, the resulting proxy is actually a complex flow that includes policies for transforming data extracting and setting variables, manipulating messages, adding threat protection, and more. After you generate a proxy based on a WSDL, take a look at the resulting flow in the Develop view of the API management UI. There, you can see exactly which policies have been added.

For example, the following figure shows the Target Endpoint `Preflow` part of a SOAP-based proxy. On the request side, an `AssignMessage` policy is used to set the target URL. On the response side, policies execute to transform the response from XML to JSON, extract the SOAP body part of the response into a variable, and set the response message. These policies (and others) are added automatically when you create the proxy.

OpenAPI Spec: To view the auto-generated OpenAPI Spec for this proxy, visit `http(s)://[proxy_domain]/[proxy_base_path]/openapi.json`. However, the conversion is not always accurate, since not all the rules of an XML schema can be represented in an OpenAPI Spec.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Note

When you face any issue with the request and response, refer the "SAP NOTE 0002115216".

3.10.16.2 Creating a Pass-through Proxy to a SOAP-Based Service

This section explains how to create a pass-through proxy with the Pass-Through Proxy option in the Create New Proxy dialog.

3.10.16.2.1 Overview

The Pass-Through Proxy option lets you to create a proxy that passes the SOAP message in a request to the backend service "untouched", making it very easy to create a proxy for a SOAP-based web service. Behind the scenes, SAP API Management handles any transformations and other flow activities for you automatically. For example, if the request happens to be in JSON format, SAP API Management takes steps to convert it to a valid XML SOAP message with correct namespaces before POSTing it to the service. Similarly, when the service returns an XML-based SOAP response, SAP API Management translates it back to JSON before returning it to the client. In addition, SAP API Management sets up the backend target endpoint, which can vary per SOAP operation.

Note:

You might wish to choose Pass-Through if the WSDL operations support a lot of unbounded parameters. It's easier for SAP API Management to translate a WSDL containing bounded parameters to a proxy, because they are finite and therefore can be represented by a finite set of query parameters or form variables.

For this type of proxy, SAP API Management Edge hosts the WSDL and creates a flow in the proxy to let you access it. The address to this SAP API Management Edge-hosted WSDL, `http(s)://[proxy_domain]/[proxy_base_path]?wsdl`, becomes the new service endpoint URL for clients calling the SOAP service through the proxy.

3.10.16.2.2 Basic Steps

1. From the Dashboard, choose + API Proxy.
2. In the Build a Proxy wizard, select SOAP service.
3. Click Next.
4. In the Details page, make these selections. You must click Validate after selecting a WSDL.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

In this field	do this
WSDL	Select the source of the WSDL. · URL - Enter the URL of the WSDL you wish to use. · File - Choose a WSDL file on your file system. In cases where there are additional dependent files, you can select all of them. · Example URL - Select from a list of WSDLs for publicly available web services. These are handy for trying out the SOAP / API proxy features of SAP API Management Edge.
Proxy Name	This is name for the proxy you're creating.
Proxy Base Path	The Proxy Base Path is a URI fragment that uniquely identifies the API that is exposed by this API proxy. API Services uses the Base Path URI to match and route incoming request messages to the proper API proxy. (The Base Path is appended to the domain of the API, which is automatically generated based on your organization name and the environment where the API proxy is deployed.) It's a best practice to include a version number in the project name, for example, `/v1/weather`. This will determine how your API is invoked by consumer apps.
Description	A brief description of the proxy.

5. Click Next.
6. In the WSDL page, select the API proxy type Pass-Through SOAP.

> **Note**
>
> A table appears listing each WSDL operation and its corresponding SOAP payload. This is the payload that is "passed through" to the backend SOAP service.

7. Select from the Port Type column which set of operations you wish to use. In WSDL, port type elements define the operations that you can call on a web service.
8. Click through the rest of the wizard to add security, select virtual hosts, and deployment environment.
9. In the Build page, click Build and Deploy. SAP API Management Edge generates and deploys the new API proxy based on the WSDL.

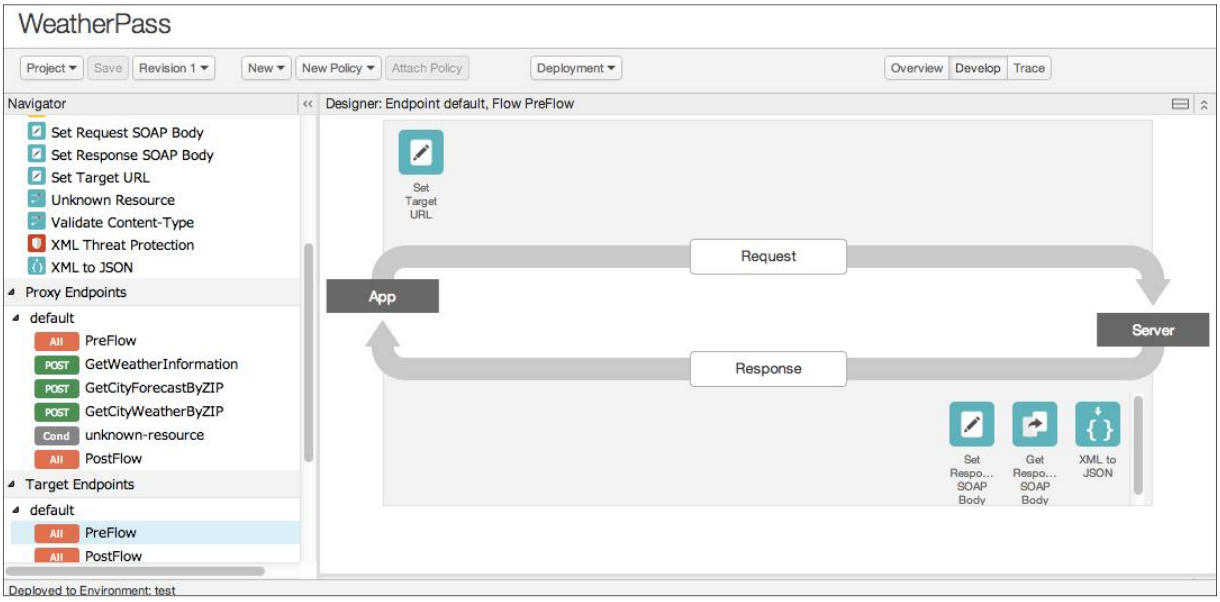
In the proxy's Overview page, the Resources list provides a detailed description of the new "pass-through" API proxy. You can think of this representation as the API's reference documentation. SAP API Management generates this view of the API model automatically for you. Simply expand a resource to see its description. The SOAP message body that is POSTed to the backend service is captured in the model view, as shown in this figure. Note that the method used for Pass-Through proxies is POST. This is the only method supported by the SOAP protocol.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.16.2.3 About the Final Proxy

When SAP API Management generates a pass-through proxy, the resulting proxy is actually a complex flow that includes policies for transforming data, extracting and setting variables, manipulating messages, adding threat protection, and more. After you generate the pass-through proxy, take a look at the resulting flow in the Develop view of the API management UI. There, you can see exactly which policies have been added.

For example, the following figure shows the Target Endpoint Preflow part of a pass-through proxy. On the request side, an AssignMessage policy is used to set the target URL. On the response side, policies execute to transform the response from XML to JSON, extract the SOAP body part of the response into a variable, and set the response message. These policies (and others) are added automatically when you create the proxy.



Note

When you face any issue with the request and response, refer the "SAP NOTE 0002115216".

SAP API Management Edge-hosted WSDL: To see the SAP API Management Edge-hosted WSDL generated for this type of proxy, go to `http(s)://[proxy_domain]/[proxy_base_path]?wsdl`.

3.10.16.3 Advanced SOAP-to-REST proxy development

The previous sections covered the creation of a SOAP-to-REST API proxy using the API proxy wizard in SAP API Management Edge. However, if you want more fine-grained control over the SOAP-to-REST transformation, you can bypass the automation provided by the wizard and build a proxy by manually adding and configuring policies to get the behavior you want.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.10.17 Streaming Requests and Responses

After reading this topic you will know how to enable streaming HTTP requests and responses to and from API proxies deployed to SAP API Management.

3.10.17.1 What is Request and Response Streaming?

By default, HTTP streaming is disabled and HTTP request and response payloads are written to a buffer in memory before they are processed by the API proxy pipeline. You can change this behavior by enabling streaming. With streaming enabled, request and response payloads are streamed without modification to the client app (for responses) and the target endpoint (for requests).

3.10.17.2 When should I enable streaming?

If your API proxy handles very large requests and/or responses, you may wish to enable streaming. For example, you may wish to enable a streaming response flow if you have a Node.js application deployed on SAP API Management that returns output over time, such as a Comet-style application. Or, you may wish to enable streaming for any application that accepts or returns a very large amount of data.

> **Note**
>
> If your API requests or returns a large amount of data, you may see this HTTP error:
>
> ```
> {"fault":"{\"detail\":{\"errorcode\":\"protocol.http.TooBigBody\"},\"faultstring\":\"Body buffer overflow\"}"}
> ```
>
> If you see this error, we recommend that you enable streaming. If you continue to see the error after you enable streaming, consider removing any policies that require access to the request or response payload.

3.10.17.3 What else should I know about streaming?

When streaming is enabled, policies that require access to the request or response payload, such as XSLT transformations and XML to JSON policies cannot be run and are bypassed.

3.10.17.4 How to enable response and request streaming

To enable request streaming you need to add the `request.streaming.enabled` property to the ProxyEndpoint and TargetEndpoint definitions in the proxy bundle and set it to true. Similarly, set the `response.streaming.enabled` property to enable response streaming.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

You can locate these configuration files in the management UI in the Develop view for your proxy. If you are developing locally, these definition files are in `apiproxy/proxies` and `apiproxy/targets`.

This sample shows how to enable both request and response streaming in the TargetEndpoint definition.

```
<TargetEndpoint name="default">
  <HTTPTargetConnection>
    <URL>http://weather.yahooapis.com</URL>
    <Properties>
      <Property name="response.streaming.enabled">true</Property>
      <Property name="request.streaming.enabled">true</Property>
      <Property name="supports.http10">true</Property>
      <Property name="request.retain.headers">User-Agent,Referer,Accept-
Language</Property>
      <Property name="retain.queryparams">apikey</Property>
    </Properties>
  </HTTPTargetConnection>
</TargetEndpoint>
```

This example shows how to enable response and request streaming in the ProxyEndpoint definition:

```
ProxyEndpoint name="default">
  <HTTPProxyConnection>
    <URL>http://weather.yahooapis.com</URL>
    <Properties>
      <Property name="allow.http10">true</Property>
      <Property name="response.streaming.enabled">true</Property>
      <Property name="request.streaming.enabled">true</Property>
    </Properties>
  </HTTPProxyConnection>
</ProxyEndpoint>
```

3.11 Deploying API Proxies

This topic provides a quick overview of proxy deployment. You can deploy proxies using the Management UI, command-line scripts, or with APIs.

3.11.1 Deploying API proxies to Environments

When you create an API proxy you'll need to decide which environment you'll be working in. You can choose to create a new API proxy on production, but that is not recommended as you may expose an API to developers before its ready. In general you want to start by creating an API proxy in test which, after testing, you then 'promote' to prod.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Recommendation

Depending on your role, you may not be able to deploy to all environments. For example, the 'user' role can only deploy to the test environment. If you're an administrator you can deploy to any environment.

For more information, see Understanding deployment.

3.11.1.1 Development in Test Environment

As you work on an API proxy, API Platform saves iterations of your configuration as revisions. When you deploy an API proxy, you choose a specific revision to deploy. Typically, you deploy the most recent revision, and, if necessary, revert to the previous revision number. You can choose where to deploy those revisions. For instance, you can promote a revision to prod to allow developers to start working with your API. At the same time, you may be iterating multiple revisions on test, where you're adding features or fine-tuning policies. Then, when you're ready, you can deploy the new revision to prod, overwriting the existing revision on that environment. Using this method, you can always have a live revision of your API available to developers while you're developing

3.11.1.2 Promotion to Prod

When an API proxy has been fully implemented and tested, it is ready to be promoted to 'prod'. The revision of the API proxy in test will be used to overwrite the revision of the API proxy deployed on prod.

API Platform provides capabilities to ensure seamless deployment of API proxies, minimizing the impact on apps and end users during the deployment procedure.

3.11.1.3 Scripting Deployment

The SAP API Management management UI enables you to deploy API proxies to prod directly from the API proxy builder. However, in many situations the requirements for security, reliability, and consistency will mandate that development teams script deployment procedures. To do so, you can write code and scripts that invoke the RESTful API exposed by API Platform.

3.11.1.4 Environment Resources

For additional control during promotion, it is recommended that you only iterate on API proxies in test, and make as few changes as necessary to API proxies deployed in prod.

To do so, you need to ensure that certain resources associated with each environment are configured in such a way that they can remain static in an API proxy configuration.

Target URLs: It is common for API proxies to call different backend URLs during testing and production. You can use TargetServer configurations to create environment-independent TargetEndpoint configurations. See Load Balancing Across Backend Servers.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Caches and Key/value maps: Both persistence resources are scoped by environment. You should ensure that naming conventions are used to enable API proxies to store data without requiring configuration changes during promotion. See Manage caches for an environment.

ServiceCallout targets: Service callouts may use different targets depending on the environment, if, for example, a ServiceCallout in the test environment consumes a demo service. See Call services or APIs using ServiceCallout.

To make API proxy configurations environment-independent, you can also use conditional statements. Conditional statement built with the environment.name variable can be used to evaluate the current environment before enforcing a policy or before routing to a URL on the backend.

For more information, see Understanding deployment.

3.11.2 Understanding Deployment

3.11.2.1 When to deploy a proxy

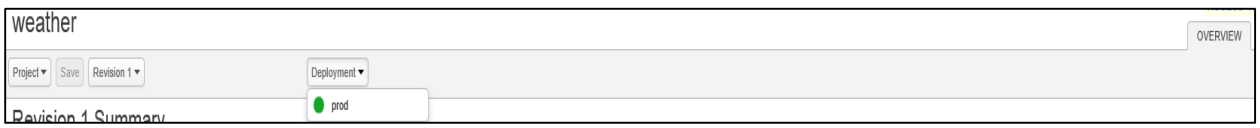
A proxy must be deployed before it can be invoked. Generally, it's up to you when you deploy. When you are working in a test environment, you may deploy iteratively many times. On the other hand, the decision to deploy a proxy from the test environment to a production environment usually depends on lifecycle rules established by your development team.

Deploy or redeploy a proxy when you:

- Create a new proxy (deployment happens automatically)
- Modify an existing proxy
- Create a new revision of a proxy
- Create a new version of a proxy
- Push a proxy from one environment to another, such as from a test environment to a production environment.
- Delete and recreate a keystore.

3.11.2.2 Where to deploy a proxy

You deploy a proxy to an environment. All organizations in SAP API Management, by default, have two environments called test and prod. These environments are merely designed to provide you with one area to work on and test API changes, and another area where APIs are exposed to apps. The following figure shows a proxy that is deployed to the test environment, as indicated by the green dot.



© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Note

Depending on your role, you may not be able to deploy to all environments. Users can only deploy to the test environment. If you're an administrator you can deploy to any environment.

3.11.2.3 How to deploy a proxy

How you deploy a proxy depends on where you are developing the proxy. If you are working in the UI, you can easily deploy a proxy with just a couple of mouse clicks. A new proxy is automatically deployed when you create it; you don't have to do anything special. The procedure for redeploying an existing proxy is almost as simple. Just select which deployment environment to deploy to, and the management UI takes care of the rest. For more information, see Deploying proxies in the UI.

If you are developing proxies and related components offline (that is, working with proxy XML files and other code directly on your file system), SAP API Management provides a convenient command-line deployment tool that you can use. You can also obtain a sample shell script that you can configure and run to upload and deploy your proxy files. For more information, see Deploying proxies from the command line.

Finally, you can use the SAP API Management API to deploy proxies. The deployment APIs exposes atomic functions that your development team can coordinate to automate and optimize your API development lifecycle. See Deploy API proxies using the management API.

3.11.2.4 What is an API Proxy Revision?

Revisions let you manage API proxy updates as you create and deploy them to an environment. Revisions are sequentially numbered, enabling you to revert changes by deploying a previous revision of your API proxy. Only one revision of an API proxy can be deployed in an environment at a given time.

Typically, an existing revision must be undeployed before a new one can be deployed. Other advanced options include overwriting a deployed revision or opting not to increment a revision at all. For example, sometimes when you make minor changes, you might not want to increment the revision. These advanced options can be accomplished through direct calls to the SAP API Management API. See Deploy API proxies using the management API.

You can deploy a revision of an API proxy into the prod environment, while continuing to create new revisions of that API proxy in the test environment. When you are ready, you can "promote" the higher revision of your API proxy from the test environment over the prior revision of the API proxy in the prod environment.

For more information on revisions, see Deploy API proxies using the management API. See also Deploying proxies in the UI.

3.11.2.5 What is an API Proxy Version?

While a revision usually represents a relatively minor update, made in the course of a development cycle, the version of an API proxy should change relatively infrequently. Typically, a version reflects a significant change in

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

the signature of an API and requires developers to modify their code accordingly. Versions are usually reflected directly in the API proxy's base URI. For example:

- api.company.com/v1/...
- api.company.com/v2/...
- api.company.com/v3/...

> **Note**
>
> Don't confuse the version of an API (that is, the public interface) with the revision of an API proxy (that is, the internal number associated with a configuration). The two numbers are unrelated, and the revision number of an API proxy is totally opaque to apps that consume your API.

As an API provider, you need to determine when it's appropriate to release a new version. For more information, see the next section, "Versioning best practices."

3.11.2.6 Versioning best practices

Keep the following best practices in mind when you consider versioning:

- Never release an API without a version.
- Make the version mandatory.
- Specify the version with a "v" prefix. Place the prefix to the far left in the URL so that it has the highest scope, e.g., /v1/dogs.
- Use a single ordinal number, such as v1, v2, and so on.
- Make the version part of the API base path
- Maintain at least one version back.
- Set up a reasonable deprecation policy. Give developers plenty of time and warning before deprecating features. Depending on your developer's platform, that can be six months or two years. For example, mobile apps take longer to rev than web apps.

> **Note**
>
> Do not use the dot notation, like v1.2 for an API proxy version. This implies a granularity of versioning that doesn't work well with APIs.

3.11.3 Deploying Proxies in the UI

All organizations in SAP API Management, by default, have two environments: test and prod. The naming is arbitrary. These environments are merely designed to provide you with one area to work on and test API changes, and another area where APIs are exposed to apps.

> **Note**
>
> Depending on your role, you may not be able to deploy to all environments. Users can only deploy to the test environment. If you're an administrator you can deploy to any environment.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.11.3.1 Deployment Options

You can deploy multiple revisions of an API. It's common to have a revision in prod while another revision is in test as it's being tested or developed. As long as the API proxy base path stays the same, each environment can only have one deployed revision of an API. For example, you could have revision 1 deployed in test and revision 20 deployed in prod. You can view deployment of all revisions on the Overview page of the API proxy editor.

When you create a new revision of an API proxy without changing its base path, then deploy it to an environment in which it's already deployed, the previous version is undeployed and the new revision is deployed in its place. Note that deployment through the management UI might impact inbound calls. To handle and transition inbound calls more gracefully during deployment, use the management API.

3.11.3.2 Deploying and Undeploying an API Revision

1. In the API proxy editor, select the revision you want to deploy.
2. Choose Deployment > {environment} and respond to the confirmation dialog.

If the API proxy base path is the same as another deployed revision in that environment, the former revision is undeployed and the new revision is deployed in its place.

For deploying multiple revisions in the same environment, see the next section.

> **Note**
>
> You can also undeploy a revision by performing the previous steps on an already deployed revision, indicated by a green dot next to the environment name.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

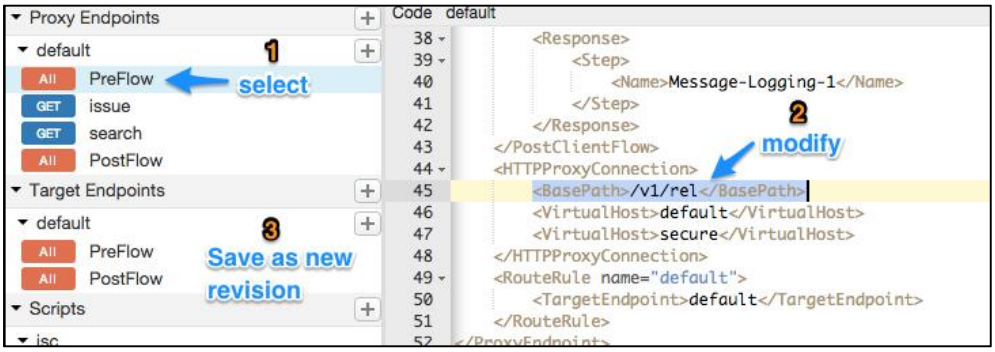
3.11.3.3 Deploying multiple revisions in the same environment

You can deploy multiple revisions of an API proxy in a single environment by changing the proxy's base path, saving the proxy as a new revision, and deploying the new revision.

For example, this is a great way to create multiple versions of an API proxy by specifying /v1, /v2, /v3, etc. in the base path. You could deploy all of those revisions to a single environment (such as test). Each proxy could be completely different, with different flows, policies, and configurations, and you could make API calls to each in the test environment.

To create and deploy a new API proxy revision by changing the base path

1. Go to the Develop page. In the Navigator, select one of the Proxy Endpoint flows (such as the default item or PreFlow).
2. Change the value of the <BasePath> element.
3. Select Project > Save as New Revision.



At this point, you can make any flow, policy, or other configuration changes you want to the API proxy and choose Save.

4. Deploy the new revision by selecting Deployment > {environment}.

On the Overview page, you can see your multiple revisions, all with different base paths, deployed to the same environment.

3.11.3.4 Running the SAP API Management Python deploy tool

The SAP API Management Python deploy tool works with the SAP API Management API to import and deploy API proxies. The tool is available as part of the API Platform Samples distribution on GitHub.

Use the SAP API Management Python tool to import (upload) and deploy your API in one simple process.

The deploy tool must be run from the base directory in the distribution of the API Platform Samples distribution, where the base directory is the parent directory of the setup and tools directory.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

In the following command, substitute the proxy name, your SAP API Management username and password, the SAP API Management organization and environment, and the path to the directory that contains the API proxy's "apiproxy" directory:

```
$python tools/deploy.py -n proxyname -u name:passW -o org -e env -d proxypath -p /
```

For example:

```
$ python tools/deploy.py -n weatherapi -u me@myCo.com:foo -o myCo -e test -d
weatherapi -p /
```

This command zips your files, pushes them to your organization on SAP API Management, and deploys them to the specified environment.

The deploy tool automatically discovers the current revision of your API proxy, undeploys the existing revision, and deploys an incremented revision of the API proxy.

On success you see output in the form:

```
Writing ./<proxybasedir>/apiproxy/weatherapi.xml to apiproxy/weatherapi.xml
Writing ./<proxybasedir>/apiproxy/proxies/default.xml to apiproxy/proxies/default.xml
Writing ./<proxybasedir>/apiproxy/targets/default.xml to apiproxy/targets/default.xml
Imported new proxy version 1
Environment: test
  Revision: 1 BasePath = /
  State: deployed
```

You can use this tool to upload the API proxy to an on-premises version of SAP API Management by specifying the -h flag:

```
$ python tools/deploy.py -n weatherapi -u me@myCo.com:foo -o myCo -e test -d
weatherapi -p / -h https://192.168.11.111:8080
```

In this example, you specify the IP address of the SAP API Management server. If you have created a DNS record for the management server, you can specify a URL in the form https://ms_URL:8080.

The complete list of flags for the command are:

- -n: The name of your API proxy.
- -u: The username and password for your account in an organization in SAP API Management.
- -o: The name of the organization in which you have an account.
- -e: The environment to which the API proxy should be deployed (test or prod).
- -d: The path to the directory containing your API proxy files. Your API proxy files must be stored under a directory named "apiproxy". This value is the path of the directory that contains the "apiproxy" directory, and not to the path to the "apiproxy" directory itself.
- -p: The URI path used as a pattern match to route incoming message to this API proxy deployment. In most cases, you can set this value to '/', unless you have advanced deployment and routing requirements. The primary path used for API proxy routing is defined in the API proxy's ProxyEndpoint configuration file.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- -h: Use the URL of the SAP API Management server and port 8080, as in: https://ms_URL:8080. Or, use the IP address of the management server, as in `https://192.168.11.111:8080`.

3.11.3.5 Configuring and running a deploy shell script

The shell script described in this section is included with the API Platform Samples distribution. This section assumes that you have obtained the shell script.

The easiest approach is to run the deploy script provided with the sample API proxy. The shell scripts wrap the SAP API Management Python deploy tool.

From the `/simplyProxy` directory run:

```
$ sh deploy.sh
```

You should see:

```
Enter your password for user {myname} in the SAP API Management Enterprise
organization {org_name}, followed by [ENTER]:
```

Enter your password, press ENTER.

You should then see:

```
Deploying to test on https://<host>:<port>/ using {myname} on <host>:<port>/ and
{org_name} on <host>:<port>/
```

If you see the following:

```
Enter your password for user Your USERNAME on enterprise.sap.com in the SAP API
Management organization Your ORG followed by [ENTER]:
```

it means that you need to modify the /setup/setenv.sh file in the platform samples distribution.

On success, the deploy tool ZIPs up the files under /apiproxy, imports the package to your organization on SAP API Management, and then deploys the API proxy to the 'test' environment.

Your API proxy is ready to be invoked.

3.11.4 Deploy API proxies using the Management API

Every organization has a unique software development lifecycle (SDLC). It is often necessary to synchronize and align API proxy deployment with the processes used for backend services.

The API methods demonstrated in this topic can be used to integrate API proxy management into your organization's SDLC. A common usage of this API is to write scripts or code that deploy API proxies, or that

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

migrate API proxies from one environment to another, as part of a larger automated process that also deploys or migrates other applications.

The management API makes no assumptions about your SDLC (or anyone else's, for that matter). Rather, it exposes atomic functions that can be coordinated by your development team to automate and optimize your API development lifecycle.

This topic focuses on the set of APIs that are for managing API proxies.

3.11.4.1 Interacting with the API

The following steps walk you through simple interactions with the APIs.

List APIs in your organization

You can begin by listing all of the API proxies in your organization. (Remember to substitute entries formyname:mypass and {org_name}.

```
$ curl -u myname:mypass http://<host:port>/v1/o/{org_name}/apis
```

Sample Response:

```
[ "weatherapi" ]
```

Get an API

You can call the GET method on any API proxy in your organization. This call returns a list of all available revisions of the API proxy.

```
$ curl -u myname:mypass -H "Accept: application/json"
http://<host:port>/v1/o/{org_name}/apis/weatherapi
```

Sample Response:

```
{
  "name" : "weatherapi",
  "revision" : [ "1" ]
}
```

The only detail returned by this method is the name of the API proxy along with the associated 'revision', which has an associated number. API proxies consist of a bundle of configuration files. Revisions provide a lightweight mechanism for managing your updates of the configuration as you iterate. Revisions are sequentially numbered, enabling you to revert changes by deploying a previous revision of your API proxy. Also, you can deploy a revision of an API proxy into the prod environment, while continuing to create new revisions of that API proxy in the test environment. When you are ready, you can 'promote' the higher revision of your API proxy from the test environment over the prior revision of the API proxy in the prod environment.

In this example, there is only one revision because the API proxy was just created. As an API proxy moves through the lifecycle of iterative configuration and deployment, the revision number increments by integers. Using direct API calls to deploy, you can optionally increment the revision number of the API proxy. Sometimes when you make minor changes, you might not want to increment the revision.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Get API Revision

> **Note**
>
> Don't confuse the version of an API (that is, the public interface) with the revision of an API proxy (that is, the internal number associated with a configuration). The two numbers are unrelated, and the revision number of an API proxy is totally opaque to apps that consume your API.

The API version (for example, api.company.com/v1) should change very infrequently. When you do increment the API version, it signifies to developers that there has been a significant change in the signature of the external interface exposed by the API.

The API proxy revision is merely an incremented number associated with an API proxy configuration. API Platform maintains revisions of your configurations so that you can revert a configuration when something goes wrong. By default, an API proxy's revision is automatically incremented every time you deploy an API proxy. This behavior can be overridden in the management UI and in the API.

To obtain the detailed profile of an API proxy configuration, you call the GET method on a specific revision number.

For example, you can call the GET method on API proxy revision 1 to get a detailed view.

```
$ curl -u myname:mypass -H "Accept:application/json"
http://<host:port>/v1/o/{org_name}/apis/weatherapi/revisions/1
```

Sample Response

```
{
  "configurationVersion" : {
    "majorVersion" : 4,
    "minorVersion" : 0
  },
  "contextInfo" : "Revision 1 of application weatherapi, in organization
{org_name}",
  "createdAt" : 1343178905169,
  "createdBy" : "andrew@yourcompany.com",
  "lastModifiedAt" : 1343178905169,
  "lastModifiedBy" : "andrew@yourcompany.com",
  "name" : "weatherapi",
  "policies" : [ ],
  "proxyEndpoints" : [ ],
  "resources" : [ ],
  "revision" : "1",
  "targetEndpoints" : [ ],
  "targetServers" : [ ],
  "type" : "Application"
}
```

These API proxy configuration elements are documented in detail in the API proxy configuration reference.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Deploying an API to an environment

Once your API proxy is configured to receive and forward requests properly, you can deploy it to one or more environments. Usually, you iterate on API proxies in 'test' and then, when ready, you 'promote' the API proxy revision to 'prod'. Often, you will find that you have many more revisions of an API proxy in the test environment, primarily because you will do much less iteration in the prod environment.

An API proxy cannot be invoked until is has been deployed to an environment. Once you have deployed the API proxy revision to prod, you can then publish the 'prod' URL to external developers.

Recommendation

As you've seen, you deploy the API revision to an environment, so the first step in deploying API proxy revisions is to verify the list of environments in your organization.

How to list environments

Every organization has at least two environments: 'test' and 'prod'. The distinction is arbitrary. The goal is to provide you with an area to verify that your API proxy is working properly before you open it up to outside developers.

Each environment is really just a network address, enabling you segregate traffic between the API proxies that you are working on, and those that are being accessed by apps at runtime.

Environments also provide segregation of data and resources. You can for example, set up different caches in test and prod, which can be accessed only by API proxies executing in that environment.

View environments in an organization

```
$ curl -u myname:mypass http://<host:port>/v1/o/{org_name}/environments
```

Sample Response:

```
[ "test", "prod" ]
```

Explore Deployments

A "deployment" is a revision of an API proxy that has been deployed in an environment. An API proxy that is in the 'deployed' state is accessible over the network, at the addresses defined in the VirtualHost for that environment.

Recommendation

It is often useful to check to see in which environment (and if!) your API is deployed. For example, you might find requests to your API proxy are failing. One troubleshooting step is to ensure that the API proxy is deployed as expected.

3.11.4.2 Deploying API Proxies

API proxies cannot be invoked until they have been deployed. API Platform exposes RESTful APIs that provide control over the deployment process.

The following APIs are called on your behalf by the Python deploy tool. The deploy tool provides support for packaging and importing API proxies that you develop locally, and it also manages undeployment of existing API

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

proxy revisions. Use the direct API calls described below when you need additional control over the deployment process.

Only one revision of an API proxy can be deployed in an environment at a given time. Therefore the deployed revision needs to be undeployed. You can control whether the new bundle is deployed as a new revision or whether it overwrites the existing revision.

First undeploy the existing revision. Using query parameters, specify the environment name and the revision number of the API proxy you want to undeploy:

```
$ curl -X DELETE \
http://<host:port>/v1/o/{org_name}/environments/{env-
name}/apis/{api_name}/revisions/{revision_number}/deployments"
-u myname:mypass
```

Then deploy the new revision. (You can deploy using the same revision number, or you can increment it.)

```
$ curl -X POST -H "Content-type:application/x-www-form-urlencoded" \
http://<host:port>/v1/o/{org_name}/environments/{env-
name}/apis/{api_name}/revisions/{revision_number}/deployments" \
-u myname:mypass
```

> **Note**
>
> Using this approach, there will inevitably be some lag between the time when the first package is undeployed and the next one is deployed. During this interval, calls from apps may be rejected with an HTTP code 5xx. If this is a problem, as it usually is in production deployments, use the seamless deployment option, described below.

Seamless Deployment

To minimize the potential for downtime during deployment, use the override option on the deployment method, and set it to true.

You cannot deploy one revision of an API proxy on top of another. The first must always be undeployed. By settingoverride to true, you indicate that one revision of an API proxy should be deployed over the currently deployed revision. The result is that the deployment sequence is reversed--the new revision is deployed, and once the deployment is complete, the already deployed revision is undeployed.

```
$ curl -X POST -H "Content-type:application/x-www-form-urlencoded" \
http://<host:port>/v1/o/{org_name}/environments/{env-
name}/apis/{api_name}/revisions/{revision_number}/deployments?"override=true" \
-u myname:mypass
```

You can further optimize deployment by setting the delay parameter. The delay parameter specifies a time interval, in seconds, before which the previous revision should be undeployed. The effect is that in-flight transactions have a time interval in which to complete before the API proxy processing their transaction is undeployed. Following is what occurs with override=true and the delay parameter set:

- Revision 1 is handling requets.
- Revision 2 is being deployed in parallel.
- When Revision 2 is fully deployed, new traffic is sent to Revision 2. No new traffic is sent to Revision 1.
- However, Revision 1 may still be processing existing transactions. By setting the delay parameter (for example, 15 seconds), you give Revision 1 15 seconds to finish processing existing transactions.
- After the delay interval, Revision 1 is undeployed.

```
$ curl -X POST -H "Content-type:application/x-www-form-urlencoded" \
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
http://<host:port>/v1/o/{org_name}/environments/{env-
name}/apis/{api_name}/revisions/{revision_number}/deployments?"override=true&delay=
15" \
-u myname:mypass
```

Query Parameter	Description
override	Default is false (normal deployment behavior: existing revision is undeployed, then new revision is deployed). Set to true to override the normal deployment behavior and provide seamless deployment. The existing revision remains deployed while the new revision is also being deployed. When the new revision is deployed, the old revision is undeployed. Use in conjunction with the delay parameter to control when undeployment occurs.
delay	To allow transaction processing to complete on the existing revision before it is undeployed—and eliminate the possibility of 502 Bad Gateway or 504 Gateway Timeout errors—set this parameter to the number of seconds you want undeployment to be delayed. There is no limit to the number of seconds you can set, and there are no performance ramifications for setting a large number of seconds. During the delay, no new traffic is sent to the old revision. Default is 0 (zero) seconds. When override is set to true anddelay is 0, the existing revision is undeployed immediately after the new revision is deployed. Negative values are treated as 0 (zero) seconds.

When override=true is used along with a delay, HTTP 5xx responses during deployment can be eliminated. This is because both API proxy revisions will be deployed simultaneously, with the older revision undeployed after the delay.

See all deployments of an API Revision

Sometimes it's necessary to fetch a list of all of the currently deployed revisions of an API proxy.

```
$ curl http://<host:port>/v1/o/{org_name}/apis/weatherapi/revision/1/deployments \
-u myname:mypass
```

```
{
  "aPIProxy" : "weatherapi",
  "environment" : [ {
    "configuration" : {
      "basePath" : "",
      "steps" : [ ]
    },
    "name" : "test",
    "server" : [ {
      "status" : "deployed",
      "type" : [ "message-processor" ],
      "uUID" : "90096dd1-1019-406b-9f42-fbb80cd01200"
    }, {
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
      "status" : "deployed",
      "type" : [ "message-processor" ],
      "uUID" : "7d6e2eb1-581a-4db0-8045-20d9c3306549"
    }, {
      "status" : "deployed",
      "type" : [ "router" ],
      "uUID" : "1619e2d7-c822-45e0-9f97-63882fb6a805"
    }, {
      "status" : "deployed",
      "type" : [ "router" ],
      "uUID" : "8a5f3d5f-46f8-4e99-b4cc-955875c8a8c8"
    } ],
    "state" : "deployed"
  } ],
  "name" : "1",
  "organization" : "org_name"
}
```

The response above contains many properties specific to the internal infrastructure of SAP API Management.

The important properties contained in the response are organization, environment, aPIProxy, name, and state. By reviewing these property values, you can confirm that a specific revision of an API proxy is deployed in an environment.

See all deployments in the test environment

You can also retrieve the deployment status for a specific environment (including the revision number of the currently deployed API proxy) using the following call:

```
$ curl -u myname:mypass
http://<host:port>/v1/o/{org_name}/environments/test/deployments
```

This returns the same result as above for every API deployed in the test environment

See all deployments in your organization

To fetch a list of all currently deployed revisions of all API proxies in all environments, use the following API method:

```
$ curl http://<host:port>/v1/o/{org_name}/deployments -u myname:mypass
```

This returns the same result as above for all API proxies deployed in all environments.

Since the API is RESTful, you can simply use the POST method, along with a JSON or XML payload, against the same resource to create an API proxy.

A profile for your API proxy is generated. The default representation of an API proxy is in JavaScript object notation (JSON). Below is the default JSON response to the POST request above, which created an API proxy called 'weatherapi'. A description of each element in the profile follows:

```
{
  "configurationVersion" : {
    "majorVersion" : 4,
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    "minorVersion" : 0
  },
  "contextInfo" : "Revision 1 of application weatherapi, in organization
{org_name}",
  "createdAt" : 1357172145444,
  "createdBy" : "you@yourcompany.com",
  "displayName" : "weatherapi",
  "lastModifiedAt" : 1357172145444,
  "lastModifiedBy" : "you@yourcompany.com",
  "name" : "weatherapi",
  "policies" : [ ],
  "proxyEndpoints" : [ ],
  "resources" : [ ],
  "revision" : "1",
  "targetEndpoints" : [ ],
  "targetServers" : [ ],
  "type" : "Application"
}
```

> **Note**
>
> Keep in mind that the response payload merely contains a representation of an API resource — you can create an API proxy using JSON or XML, and you can also retrieve representations of the API proxy as XML or JSON, depending on your needs.

The API proxy profile that is generated demonstrates the complete structure of an API proxy:

- `APIProxy revision`: The sequentially numbered iteration of the API proxy configuration, as maintained by API Platform
- `APIProxy name`: The unique name of the API proxy
- `ConfigurationVersion`: API Platform version to which the API proxy configuration conforms
- `CreatedAt`: Time when the API proxy was generated, formatted in UNIX time
- `CreatedBy`: Email address of the user who created the API proxy
- `DisplayName:` A user-friendly name for the API proxy
- `LastModifiedAt`: Time when the API proxy was updated, formatted in UNIX time
- `LastModifiedBy`: Email address of the user who updated the API proxy
- `Policies`: A list of policies that have been added to this API proxy
- `ProxyEndpoints`: A list of named ProxyEndpoints
- Resources: A list of resources (JavaScript, Python, XSLT) available to be executed in this API proxy
- `TargetServers`: A list of named TargetServers (that can be created using the management API), used in advanced configurations for load balancing purposes
- `TargetEndpoints`: A list of named TargetEndpoints

Note that many of the elements of the API proxy configuration created using the simple POST method above are empty. In the following topics, you will learn how to add and configure the key components of an API proxy.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

You can also read about these configuration elements in the API proxy configuration Reference.

3.11.4.3 Security Guidelines

The SAP Management API relies on HTTP Basic Authentication. You need to provide a username and password for each API call.

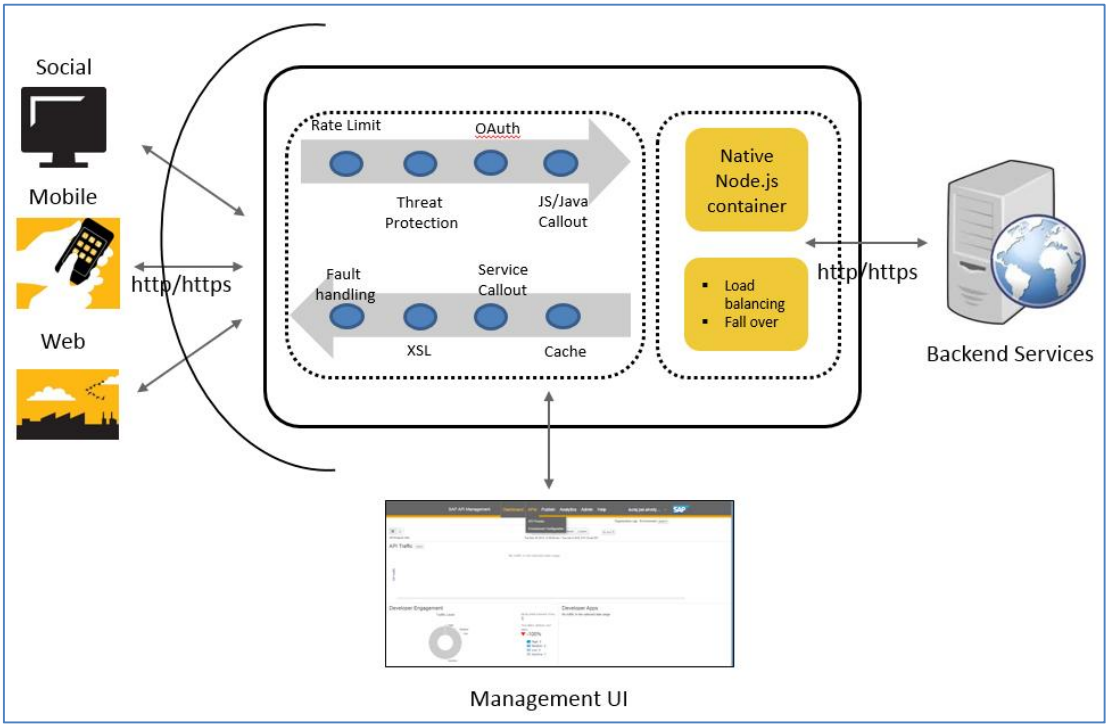
In some situations, it is not practical to collect the password when the script runs. For example, you may need to run a cron job that fires when no administrators are present. In these situations, you need to make the password available to the script without any human intervention.

Follow these guidelines:

1. Centralize credentials in a single file that is used as a source for the programs and scripts that you write
2. Protect the credentials source file to the extent possible using file system security and permissions
3. Create an automation client with highly restricted permissions on specific resources in your organization.

3.12 Building Proxies with Node.js

Node.js adds more programmability to the API platform. Running on SAP API Management, Node.js apps take advantage of SAP API Management's enterprise-grade operations like traffic management, security, deployment tools, revision control, logging, and analytics. Furthermore, you can leverage thousands of third-party Node.js modules in your APIs.



CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Enterprise developers have found many creative ways to integrate Node.js applications into the API platform. Some common use cases include:

- Build highly customized standalone APIs and backend services.
- Solve complex orchestration and mobile optimization problems using SAP API Management policies with the advantage of a scriptable target endpoint.
- Build composite services and mashups connecting to different SAP and non-SAP backends.
- Rapidly develop prototypes of new APIs.

As you explore our set of Node.js topics, you will learn how to create API proxies with Node.js and how to deploy your existing Node.js application in SAP API Management.

3.12.1 Getting started with Node.js

This topic explains the simplest way to wrap a Node.js application in an API proxy using the management UI.

Chances are, the first Node.js app you ever created was an HTTP server that responds to requests with "Hello World!" The simplest way to try out Node.js is to do something very similar. With just a few mouse clicks, you'll have a functioning, proxied Node.js HTTP server running on SAP API Management. After that, you can use the code editor in the UI to modify the Node.js app, add additional Node.js files, add policies, and so on.

3.12.1.1 Creating the sample Hello World! API proxy

This section explains how to create an API proxy that interacts with a Node.js HTTP server. The Node.js HTTP server code is already written for you and is automatically deployed to SAP API Management when you follow these simple steps.

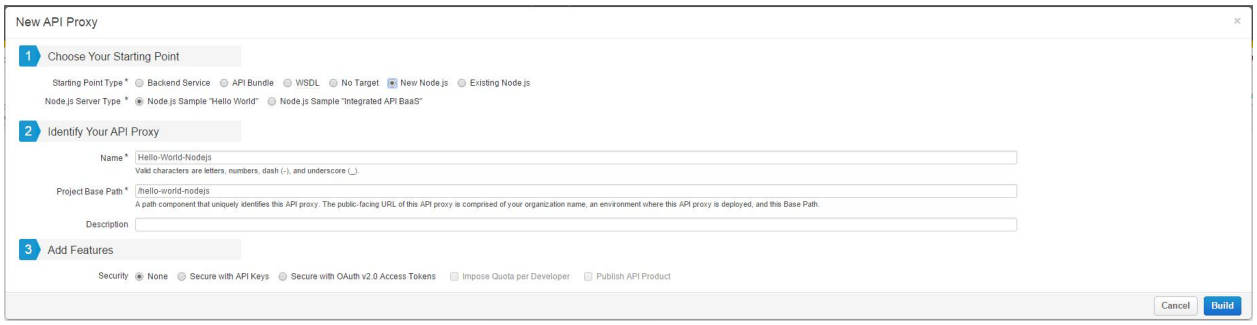
1. Log in to the SAP API Management UI.
2. From the APIs menu, select API Proxies.
3. In the API Proxies summary page, click +API Proxy.
4. In the New API proxy dialog, select New Node.js.

   Note

   Use the Node.js Sample "Integrated API BaaS" option only when SAP API Management's API BaaS component is installed.
5. From the Node.js Server Type options, select Node.js Sample "*Hello World*".
6. Provide a Name for the proxy. You can just enter *Hello*.
7. Provide a Project Base Path. If you specified *Hello* as the display name, then the path */hello* is populated automatically.
8. Add a version number **/v1** to the path. API versioning is a best practice. The new Project Base Path is:

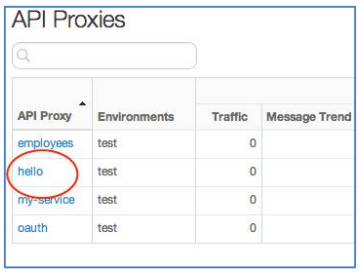
   /v1/hello
9. Optionally, provide a Description.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.



10. Choose Build.

    SAP API Management builds the proxy. When the build completes, the hello proxy appears in the API Proxies summary page. Select API Proxies from the APIs menu to see the API Proxies summary page, shown below:



3.12.1.2 Invoking the hello proxy

When you call the `hello` proxy, the Node.js application executes automatically, responding with "Hello, World!". Note that unless you specified otherwise, the Node.js application is deployed to the environment called test. Here's the basic call using Curl (substitute your hostname and port).

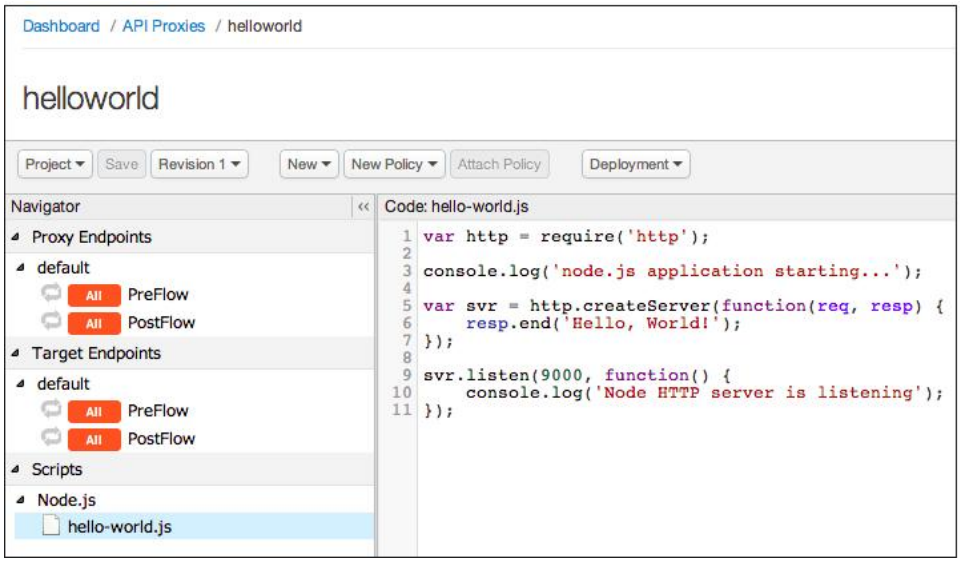
```
$ curl http://{hostname}:{port}/v1/hello
Hello, World!
```

3.12.1.3 Viewing and editing the Node.js code

Let's look at the Node.js code that was added to the API proxy. Go to the summary page for the Hello World proxy and choose Develop.

This opens the Develop view which includes a code editor. You can edit the code there directly.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.



For example, change the response from `Hello, World!` to something else, like `Hello, Node!`, then choose Save. The proxy is saved and redeployed.

Finally, re-invoke the proxy to verify the change:

```
$ curl http://{hostname}:{port}/v1/hello
Hello, Node!
```

Like all Node.js applications, Node.js applications running on SAP API Management run in a single thread of control. There is no need (and in fact no ability) to start another thread, or synchronize variables between threads. Since Node.js enforces non-blocking programming, a single script can support thousands of concurrent requests because the script gives up the CPU whenever it has to wait for something, and it is notified later when it happens.

To use Node.js on SAP API Management, you need to specify a main Node.js script file. This script must be configured to handle incoming requests, which you typically do by using the http or https modules, creating a client, and so on. (If the main script is not configured this way, it will simply execute and exit after it is deployed.) Within SAP API Management, each Node.js application script is started from the beginning when the proxy is deployed, and stopped when the proxy is undeployed. In between it will wait for new requests and process them. For more information, see "Invoking an imported Node.js file" in Deploying a standalone Node.js app.

3.12.2 Deploying a Standalone Node.js app

This topic explains how to deploy a Node.js application from your local system to SAP API Management. In this topic, we'll discuss how to use a command-line tool called SAP API Managementtool to deploy Node.js applications to SAP API Management.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.12.2.1 About deploying Node.js code to SAP API Management

You can deploy an existing Node.js API application, including any dependent Node.js modules, from your local system to SAP API Management using a command-line utility called SAP API Managementtool. The utility automatically bundles the application and its dependencies into an API proxy and deploys it on SAP API Management.

For example, let's say you used Express to create a web application in Node.js. The application runs as an HTTP server that listens for HTTP requests, processes those requests, returns data, and so on. When you use `SAP API Managementtool` to deploy a Node.js application to SAP API Management, it's wrapped in a proxy and executes within the context of the API platform. You can then call your application through its new proxy URL, and you can add value to it by "dressing it" with standard SAP API Management features like OAuth security, quota policies, threat protection policies, conditional flows, caching, and many others.

3.12.2.2 What does SAP API Management tool do?

When you run the SAP API Managementtool utility with the deploynodeapp option, it:

- Generates an API proxy configuration bundle to house the Node.js application.
- Packages the Node.js application with any Node.js packages installed with NPM (Node Packaged Modules).
- Imports the API proxy configuration bundle to the specified organization on SAP API Management using the SAP API Management API.
- Deploys the API proxy to an environment.
- Executes the Node.js application on SAP API Management and making it available over the network.

3.12.2.3 Preparing to use SAP API Management tool

Before you begin, you need to install the SAP API Management tool utility.

You can install SAP API Management tool through npm.

3.12.2.4 Installation from npm

The `SAP API Management tool` module and its dependencies are designed for Node.js and is available through npm using the following command:

```
$ sudo npm install -g apigeetool
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Note

sudo may be required with the `-g` option which places the SAP API Management tool command-line commands in you PATH. If you do not use `-g`, then you need to add the apigeetool/bin directory to your PATH manually.

Typically, the -g option places modules in: `/usr/local/lib/node_modules/apigeetool` on *nix-based machines.

3.12.2.5 Deploying a Node.js application to SAP API Management with SAP API Management tool

To deploy a Node.js application using SAP API Management tool:

1. In a terminal window, `cd` the root directory of your Node.js application.
2. Execute the `SAP API Management tool` utility with the deploynodeapp command:

   ```
   $ apigeetool deploynodeapp -n {A name for your new proxy} -d {The root directory of your Node.js app} -m {The name of the main Node.js file} -o {Your org name on SAP API Management} -e {The environment to deploy to} -b {The base URL for your proxy} -u {Your username on SAP API Management} -p {Your SAP API Management password} -L {Management Server hostname and port}
   ```

   For example:

   ```
   $ apigeetool deploynodeapp -n myNodeApp -d . -m server.js -o myorg -e test -b myNodeApp -u ntesla -p myPassword -L Management Server hostname and port
   ```

3. Check the output in your terminal window. It will look something like this:

   ```
   Importing new application myNodeApp
   Imported new app revision 1
   Deploying revision 1
     Deployed.
   Proxy: "myNodeApp" Revision 1
     Environment: test BasePath: /myNodeApp
     Status: deployed
   ```

   If you see "`Status: deployed`", that's it. Everything went smoothly. Your Node.js app is wrapped in an API proxy, deployed to SAP API Management, and it's running and waiting to handle requests. It's ready to be tested.

4. Test your proxy.

   Example

   ```
   $ curl http://{hostname}:{port}/myNodeApp
   Hello, My Node App!
   ```

   If you wish, log in to you're your SAP API Management account and go to the API Proxies page of the management UI. You will see your new proxy listed there.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.12.2.6 How SAP API Management tool handles dependent files and modules

If your Node.js application depends on installed modules, SAP API Management tool handles them by zipping the node_modules folder and adding it to the proxy bundle. No additional action is necessary. The same is true for any directories that contain additional source code. The SAP API Management tool utility zips them and deploys them with the bundle.

You cannot edit files in these zipped directories on the SAP API Management UI's editor. If you need to change them, you can export your project, edit the files locally, and then redeploy using SAP API Management tool or by importing the exported project using the management UI. See also "Exporting and importing a proxy with Node.js code".

3.12.2.7 Basic usage information for SAP API Management tool

For basic usage information on the SAP API Management tool utility's input parameters, enter:

```
$ apigeetool deploynodeapp -h
  Usage: deploynodeapp -n [name] -o [organization] -e [environment]
  -d [directory name] -m [main script file]
  -u [username] -p [password]
  -b [base path] -l [Management server url] -z [zip file] -i -h
  -o SAP API Management organization name
  -e SAP API Management environment name
  -n SAP API Management proxy name
  -d SAP API Management proxy directory
  -m Main script name: Should be at the top level of the directory
  -u SAP API Management user name
  -p SAP API Management password
  -b Base path (optional, defaults to /)
  -L Management Server hostname and port
  -z ZIP file to save (optional for debugging)
  -i import only, do not deploy
  -h Print this message
```

3.12.2.8 Creating a new proxy with existing Node.js files

Another way to integrate an existing Node.js application into an API proxy is to add the application when you create the proxy. You can do this entirely through the management UI and the New API Proxy dialog.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

This is currently the only way to incorporate an existing local Node.js application into a proxy from the management UI. This procedure must be performed when you are creating a new proxy.

1. From the API proxy summary page, choose +API Proxy.
2. In the New API Proxy dialog, select Existing Node.js.
3. Use the Choose Files button to select one or more Node.js files to import.

   > **Note**
   >
   > If you upload multiple Node.js files, you need to specify which one is the main file. (The main file is the one that you would invoke from the command line with the node command.)

4. Give the proxy a name. In this example, we are calling it hellonode.
5. Add the version /v1 to the Project Base Path. Versioning your API is a best practice.
6. Choose Build.
7. Choose Develop to enter the Develop view.
8. Open the `TargetEndpoint` file in the code editor.
9. Be sure the **`<ScriptTarget>`** element specifies the main Node.js file, as follows:

```
<ScriptTarget>
        <ResourceURL>node://server.js</ResourceURL>
        <Properties/>
</ScriptTarget>
```

10. Choose Save.

3.12.2.9 Adding and invoking new Node.js resource files

Another way to add Node.js code to a proxy is to add it directly, either through the UI or by uploading it from your local filesystem. You can also specify which Node.js file is the main file, which is the file invoked when the proxy is deployed.

Adding new Node.js files through the UI

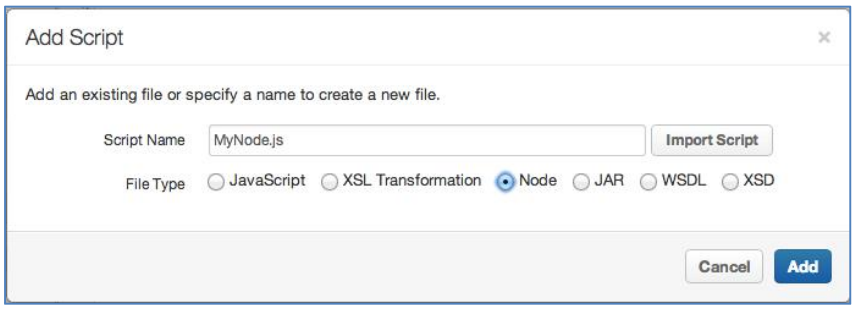
The management UI lets you add additional Node.js source files to a proxy that is on the API platform. You can create them directly in the UI or import them from your file system. First, let's look at how to do this from the UI.

> **Note**
>
> Only one Node.js source file is designated to be the main file. The main file is the one you would execute from the command line to run a Node.js application. The main file is specified in the proxy's target endpoint definition file using an element called `<ScriptTarget>`, as explained in Invoking an imported Node.js file and in Adding Node.js to an existing API proxy. In these out-of-the-box examples, the correct target endpoint definition is created for you.

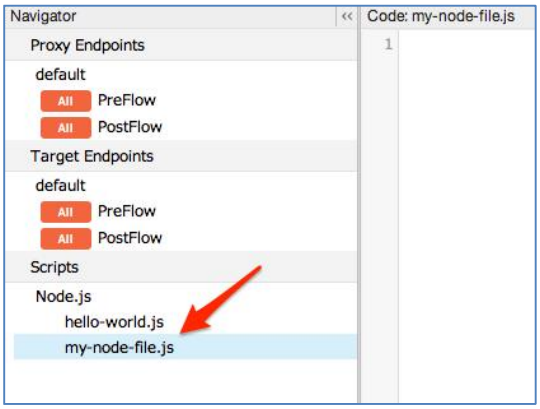
To create a new Node.js resource file:

1. In the Develop view, select New Script from the New menu.
2. In the Add Script dialog, select the file type Node and name the script.
3. Choose Add.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.



The new, blank Node.js file opens in the editor. You can cut and paste your code into the file. The file also appears in the Scripts section of the Navigator.



Importing Node.js files from your file system

To import a Node.js file from your filesystem to the proxy:

1. In the Develop view, select New Script from the New menu.
2. In the Add Script dialog, choose Import Script.
3. Use the file tool to select your Node.js file.
4. The file's name is added to the dialog, but you can change it if you wish.
5. Choose Add. The file appears in the Scripts section of the Navigator and opens in the editor.
6. Choose Save.

   If you want to invoke the imported file, an extra step is required, as explained in the next section.

Invoking an imported Node.js file

You can't simply invoke a newly imported or created Node.js file. The reason is that SAP API Management requires that one Node.js file be the main file. The main file is specified in the **`<ScriptTarget>`** element of the Target Endpoint definition. To specify which file is the main Node.js file, do the following:

1. Under Target Endpoints in the Navigator, choose the name of the target endpoint (usually called default).
2. In the Code editor, edit the `<ScriptTarget>` element by changing the `<ResourceURL>` to reflect the name of the file you wish to be the main Node.js file. For example, if you wanted a file called `hello-world.js` to be the main file, you'd enter: `node://hello-world.js` in the ResourceURL element.
3. Choose Save.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

At this point, you can invoke the file with whatever proxy path you used before. For example, we've been looking at the Hello World! Example, where the base path `v1/hello`is specified. However, you can change the base path by editing the Proxy Endpoint.

> **Note**
>
> Recall that the Proxy Endpoint defines how clients call your API. When a client calls the API with the new base path, processing is routed to the Target Endpoint, where the main Node.js file is specified.

1. Under Proxy Endpoints in the Navigator, choose the name of the proxy endpoint (usually called default).
2. In the Code editor, edit the `<HttpProxyConnection>` element by changing the <BasePath> to whatever name you wish. For example, if the current `<BasePath>` is `v1/hello` and you wanted it to be `v1/my-node-file`, change the `<BasePath>` element like this:

   ```
   <BasePath>/v1/my-node-file</BasePath>
   ```
3. Choose Save.
4. Invoke the proxy using the new base path, like this:

   ```
   $ curl http://{hostname}:{port}/v1/my-node-file
   Hello, World!
   ```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.12.2.10 Exporting and Importing a Proxy with Node.js Code

After you deploy a proxy containing Node.js code to SAP API Management, you can always export the proxy back to your system, work on it there, and then re-import it back using the management UI. This sort of round-trip development technique is commonly used.

1. From the API proxy summary page, choose Develop.
2. In the Develop page, select Download Current Revision.
3. Unzip the downloaded file on your system.

> **Note**
>
> The Download function does not unzip any directories that were zipped when the proxy was deployed, like `node_modules` or other folders with source code in them. You'll need to unzip them manually once the proxy is downloaded to your system.

You can import the proxy bundle back into SAP API Management by selecting Import Into New Revision from the same menu.

Using the Trace Tool

The trace tool is useful for general proxy debugging. For detailed information on using the trace tool, see Using the Trace tool.

Printing Console Output

You can embed console.log statements in your Node.js code and view the output in the trace tool. For example, the following statement prints the username of the user who is logging in to access the proxy:

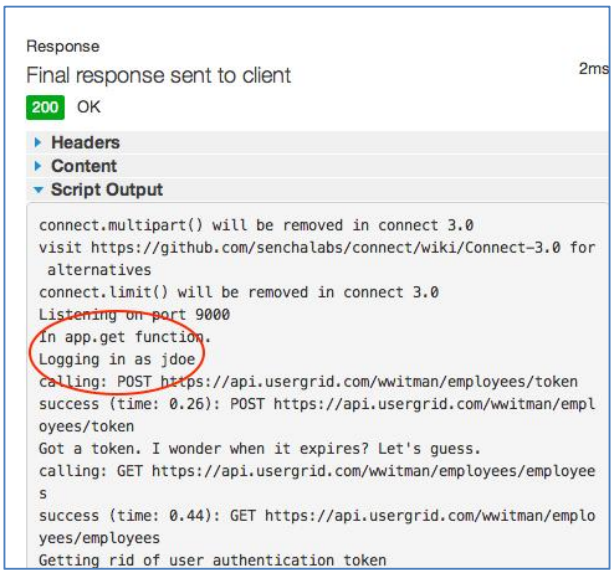
```
console.log('Logging in as %s', config.username);
```

Code: server.js

```
114         });
115     });
116 }
117
118 // We need this for UserGrid authentication
119
120 function logIn(req, res, next) {
121
122     console.log('Logging in as %s', config.username);
123
124     ug.login(config.username, config.password, function(err) {
125         if (err) {
126           console.log('Login failed: %s', JSON.stringify(err));
127           res.jsonp(500, {error: err});
128           return;
129         }
130
131         loggedIn = new usergrid.client({
132             'orgName' :  config.organization,
133             'appName' :  config.application,
134             'authType' : usergrid.AUTH_APP_USER,
135             'token':     ug.token,
136             logging:     config.logging
137         });
```

You can view the output of this console.log message in the trace tool. Simply call your API in the trace tool and open the Script Output panel of the final response, as shown below.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.



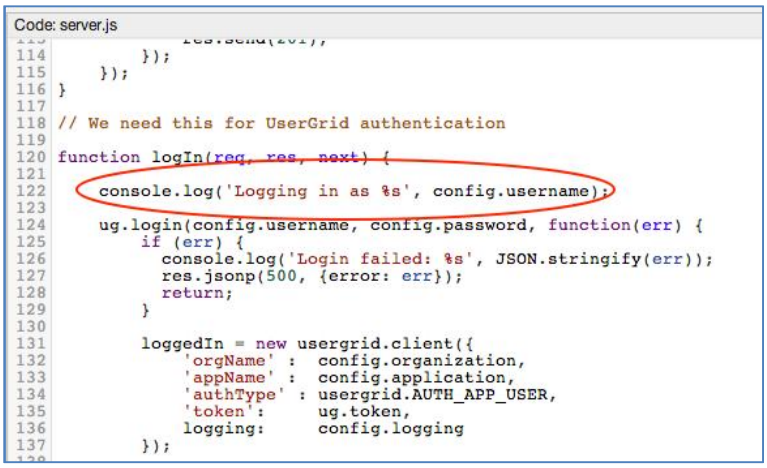
3.12.2.11 Using the Trace Tool

The trace tool is useful for general proxy debugging. For detailed information on using the trace tool, see Using the Trace tool.

3.12.2.12 Printing Console Output

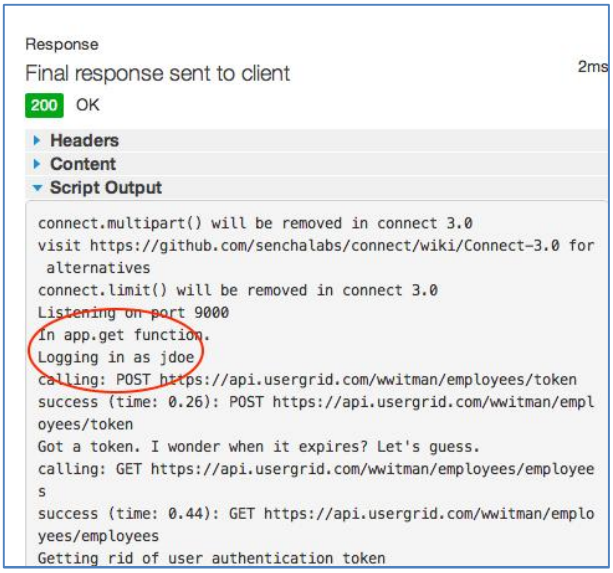
You can embed console.log statements in your Node.js code and view the output in the trace tool. For example, the following statement prints the username of the user who is logging in to access the proxy:

```
console.log('Logging in as %s', config.username);
```



© 2014 SAP SE or an SAP affiliate company. All rights reserved.

You can view the output of this console.log message in the trace tool. Simply call your API in the trace tool and open the Script Output panel of the final response, as shown below.



3.12.3 Understanding Support for Node.js Modules

Which standard Node.js modules are supported?

Use the following table to determine which standard Node.js modules are either supported, not supported, or partially supported in SAP API Management.

`Assert`	Supported	
`Buffer`	Supported	
`child_process`	Restricted	An exception will be thrown if an attempt is made to spawn a sub-process. However, "fork" is supported for spawning sub-scripts.
`cluster`	Disabled	The method cluster.isMaster always returns true, and other methods are not implemented. See Why the cluster module is disabled.
`Crypto`	Partial support	Random and HMAC functionality is supported. The remainder is unimplemented.
`Dns`	Partial support	dns.lookup is supported. All else is not implemented.
`Domain`	Supported	
`Dgram`	Restricted	Node.js applications in the SAP API Management environment will not be able to access services on

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

		the Internet via UDP due to our network architecture.
`Events`	Supported	
`Fs`	Restricted	Filesystem access is restricted to the directory where the script was launched: the `/resources/nodedirectory`. Node.js scripts may read and write files within this directory, for instance as a temporary scratch area, but there are no guarantees as to how long the files will persist.
`http`	Supported	The virtual host and path for incoming requests is specified in the API Proxy, not by the HTTP module. See "Understanding support for the http and https modules" for more information.
`https`	Supported	Creating an "https" server behaves identically to an "http" server. See "Understanding support for the http and https modules" for more information.
`module`	Supported	
`net`	Restricted	Attempts to listen for incoming TCP connections will generate an exception.
`path`	Supported	
`module`	Supported	
`process`	Partial support	Functionality to manipulate user ID, group membership, and working directory is not supported.
`punycode`	Supported	
`querystring`	Supported	
`readline`	Disabled	There is no standard input for scripts running on SAP API Management.
`repl`	Disabled	There is no standard input for scripts running on SAP API Management.
`module`	Supported	
`STDIO`	Supported	Standard output and error are routed to a log file within the SAP API Management infrastructure. There is no standard input for scripts running on SAP API Management. However, you can pass arguments using the ScriptTarget element of TargetEndpoint. See Advanced ScriptTarget configuration for more information.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

`stream`	Supported	
`string_decoder`	Supported	
`timers`	Supported	
`tls`	Supported	Transport Layer Security (TLS) parameters work basically the same way they work in regular Node.js. See Using the TLS (SSL) Node.js module on SAP API Management for details.
`tty`	Disabled	There is no standard input for scripts running on SAP API Management.
`url`	Supported	
`util`	Supported	
`vm`	Supported	
`zlib`	Supported	

3.12.4 Understanding support for the http and https modules

All Node.js applications running in SAP API Management must use the http or https module to listen for incoming requests. If you were to deploy a script that doesn't listen for incoming requests, it would simply execute and exit.

The listen method of the http and https modules in Node.js takes a port number as a parameter. For example:

```
svr.listen(process.env.PORT || 9000, function() {
   console.log('The server is running.');
});
```

This "port" argument is required in Node.js, but SAP API Management ignores this parameter. Instead, the API proxy in which the Node.js script runs specifies the "virtual host" that it listens on, and the Node.js application uses those same virtual hosts, just like any other SAP API Management proxy.

Every environment in SAP API Management has at least one virtual host. The virtual host defines the HTTP settings for connection with the SAP API Management organization. All API proxies in an environment share the same virtual hosts. By default, two virtual hosts are available for each environment: default and secure.

The SAP API Management tool deploynodeapp command generates an SAP API Management proxy wrapper around the Node.js application. When deployed, the Node.js application listens on the default virtual host defined for the environment.

Handling incoming requests

Like other SAP API Management applications, if the proxy application is set up to listen on the secure virtual host, then it will accept incoming requests using HTTPS.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Handling outgoing requests

In addition to receiving incoming traffic, Node.js applications inside SAP API Management may use the http and https modules to make outbound requests like any other Node.js application. These modules work just as they always do inside Node.js.

3.12.5 Understanding support for the tls module

SAP API Management supports the Node.js tls module. This module uses OpenSSL to provide Transport Layer Security (TLS) and/or Secure Socket Layer (SSL) encrypted stream communication. You can use the tls module to create secure connections to backend services from Node.js applications running on SAP API Management.

To understand how the tls module works on SAP API Management, it's important to understand how virtual hosts are used on SAP API Management. Every environment in SAP API Management has at least one virtual host. The virtual host defines the HTTP settings for connection with the SAP API Management organization. All API proxies in an environment share the same virtual hosts. By default, two virtual hosts are available for each environment: default and secure.

Now, let's look at how SAP API Management handles TLS (SSL) communication for incoming and outgoing requests on Node.js applications:

Handling incoming requests

Depending on how virtual hosts are configured for your organization, SAP API Management provides these options:

If the API proxy is configured to listen on the default virtual host, then it accepts requests over HTTP.

Handling outgoing requests

You can make outgoing requests with the tls module the same way you would normally in Node.js. Basically, you need to add client-side keys and certificates (.pem files) to the resources/node directory and load them inside your script. For information on using the tls module and its methods, see the Node.js tls module documentation.

3.12.6 Why the cluster module is disabled

SAP API Management does not support the "cluster" module, which allows Node.js applications to control when additional copies of an application are started and stopped, and to communicate with them. In the parlance of the "cluster" module, all Node.js applications are considered to be the "master."

However, SAP API Management does deploy multiple instances of each application — two in most cases, although the system may be configured to run more if necessary.

That means that Node.js applications running inside SAP API Management, like all Node.js applications, must assume that they will be processing many different requests "in parallel" using a single thread of control. However, the application must not assume that it is the only copy running in the world, though it also must not assume that there is some way to communicate with the other copies.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.12.7 Advanced ScriptTarget Configuration

In the <TargetEndpoint> definition, the <ScriptTarget> element takes additional optional parameters besides <ResourceURL>. You can also pass command-line arguments and environment variables to a Node.js script using the <EnvironmentVariables> and <Arguments> parameters:

```
<TargetEndpoint name="default">
  <ScriptTarget>
     <ResourceURL>node://hello.js</ResourceURL>
     <EnvironmentVariables>
         <EnvironmentVariable name="NAME">VALUE</EnvironmentVariable>
     </EnvironmentVariables>
     <Arguments>
         <Argument>ARG</Argument>
     </Arguments>
  </ScriptTarget>
</TargetEndpoint>
```

3.12.8 Using the SAP API Management-access module

The SAP API Management-access module lets you access API proxy flow variables and caches from within Node.js application code.

3.12.8.1 Obtaining SAP API Management-access

The SAP API Management-access module is integrated into the SAP API Management platform. When you deploy Node.js code to SAP API Management, this module is available to you. You simply need to require it in any Node.js code that you deploy. For example:

```
var access=require('sap-access');
```

You can also download SAP API Management-access for local development and testing. The SAP API Management-access packaged module is available through npmjs.org.

3.12.8.2 Accessing flow variables

When you deploy a Node.js application to SAP API Management, you can access any of the supported "out-of-the-box" flow variables, flow variables created by policies, and any flow variables that you create yourself from within your Node.js code. Flow variables are created and exist within the context of an API proxy running on SAP API Management.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.12.8.2.1 Methods

getVariable

```
var result = getVariable(httpRequest, name);
```

Gets a named variable.

Parameters:

- httpRequest: The request object that comes from the http module.
- name: (String) The name of the variable to retrieve.

Returns:

A string or a number, depending on the type that was set using setVariable(), when it was created by you elsewhere, or when a policy created it. If you are accessing one of the out-of-the-box SAP API Management variables, you can find a list of types in the Variables Reference. For variable types created by policies, refer to the specific policy reference topic.

Example

```
var sap = require('sap-access');
   // "httpRequest" must be a request object that came from the http module
   var val1 = sap.getVariable(request, 'TestVariable');
   var val2 = sap.getVariable(request, 'request.client.ip');
```

setVariable

```
setVariable(httpRequest, name, value);
```

Sets a variable. Some variables are read-only, and the setVariable() method throws an exception if you try to set one of them. To determine which variables are read-only, see the Variables Reference.

Parameters:

- `dhttpRequest`: The request object that comes from the http module.
- `name`: (String) The name of the variable to retrieve.
- `value`: Can be a number, String, boolean, null, or undefined.

Example

```
var sap = require('sap-access');
   sap.setVariable(request, 'TestVariable', 'bar');
   // This will throw an exception because client.ip is read-only.
   sap.setVariable(request, 'client.ip');
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

setIntVariable

```
setIntVariable(httpRequest, name, value);
```

The `setIntVariable()` method is a convenience method that first coerces the value parameter to an integer, and then sets it.

Parameters:

- `httpRequest`: The request object that comes from the http module.
- `name`: (String) The name of the variable to set.
- `value`: The value parameter must be a string or number.

Example

```
var sap = require('sap-access');
// Convert "123" to an integer and set it
sap.setIntVariable(request, 'TestVariable', '123');
// Use something that's already a number
sap.setIntVariable(request, 'TestVariable2', 42);
```

deleteVariable

Deletes a named variable. It is an error to delete a read-only variable. For a complete list of read-only variables, see the Variables Reference.

```
deleteVariable(httpRequest, name);
```

Parameters:

- `httpRequest`: The request object that comes from the http module.
- `name`: (String) The name of the variable to delete.

Example

```
sap.deleteVariable(request, 'TestVariable');
    // This will throw an exception because client.ip is a read-only variable.
    sap.deleteVariable(request, 'client.ip');
```

3.12.8.3 Accessing the cache

The sap-access module lets you access the SAP API Management distributed cache from your Node.js code.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.12.8.3.1 Methods

getCache

```
var cache = sap.getCache('cache-name');
```

Look up a named cache and create it if necessary. The resulting cache uses a pre-defined set of configuration parameters suitable for most situations.

Parameters:

cache-name - The name of the cache to access.

Returns:

A cache object.

Example

```
var sap = require('sap-access');
    var cache = sap.getCache('cache');
```

getCache

```
var customCache = sap.getCache('cache-name', { parameter: 'value'} );
```

Access a custom cache resource specified in a configuration object. For information about how to create a cache resource, see Manage Caches for an Environment.

Parameters:

cache-name - The name of the custom cache.

{parameter: 'value'} - (Optional) A configuration object. The object can be empty, or it can contain the following optional parameters:

- `resource`: The name of an SAP API Management "cache resource" where the cache data is stored. Cache resources are used to fine-tune memory allocation and other cache parameters. If not specified, a default resource will be used. If the cache resource does not exist, then the method throws an error.

- `scope`: Specifies whether cache entries are prefixed to prevent collisions. Valid values are global, application, and exclusive.

  - `global`: All cache entries may be seen by all Node.js applications in the same SAP API Management "environment."
  - `application`: All cache entries may be seen by all Node.js caches that are part of the same SAP API Management application.
  - `exclusive`: (Default) Cache entries are only seen by Node.js caches in the same application that have the same name. This is the default.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- `defaultTtl`: Specifies the default time to live for a cache entry, in seconds. If not specified then the default TTL in the cache resource will be used.
- `timeout`: How long to wait to fetch a result from the distributed cache, in seconds. The default 30 seconds. Latency-sensitive applications may wish to reduce this in order to prevent slow response times if the cache infrastructure is overloaded.

Returns:

A custom cache object.

Example

```
var sap = require('sap-access');
var customCache = sap.getCache('MyCustomCache',
  { resource: 'MyCustomResource'} );
```

put

`cache.put('key', data, ttl, function(error));`

Put data into a cache.

Parameters:

- *key*: (Required) A string that uniquely identifies the item in the cache.
- *data*: (Required) A string, Buffer, or object that represents the data to cache. Any other data type will result in an error. For convenience, objects will be converted into a string using "JSON.stringify".
- *ttl*: (Optional) The maximum time to persist the data in the cache, in seconds. If not specified then a default TTL will be used.
- *callback*: (Optional) If specified, a function that will be called once the data is safely in the cache. It will be called with an Error object as the first parameter if there is an insertion error, and otherwise it will be called with no parameters.

Example

```
var sap = require('sap-access');
var cache = sap.getCache();
// Insert a string with a timeout of 120 seconds
cache.put('key2', 'Hello, World!', 120);
// Insert a string and get notified when insert is complete
cache.put('key4', 'Hello, World!', function(err) {
  // "err" will be undefined unless there was an error on insert
});
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

get

cache.get('**key**', function(error, data));

Retrieve data from a cache.

Parameters:

- *key* (required): A string that uniquely identifies the item in the cache.
- *callback* (required): A function that will be called when the data is available. If there is cached data, it is returned in the second parameter.
  - *error* - An Error object. If there is an error while retrieving from the cache, then an Error object will be set here. Otherwise this parameter will be set to "undefined".
  - data - The data retrieved, if any. It will be one of four values:
    - If a string was inserted, it will be a string.
    - If a Buffer was inserted, it will be a Buffer.
    - If an object was inserted, it will be a string containing the JSON version of the object as produced by "JSON.stringify".
    - If nothing was found, then it will be "undefined".

Example

```
var sap = require('sap-access');
var cache = sap.getCache();
cache.get('key', function(err, data) {
  // If there was an error, then "err" will be set
  // "data" is the item that was retrieved
  // It will be a Buffer or a String depending on what was inserted.
});
```

remove

cache.remove('*key*', function(error));

Invalidate a cached item. Once a key is invalidated, subsequent get() requests will return "undefined" unless another value is inserted.

Parameters:

- `key` (Required): A string that uniquely identifies the item in the cache to invalidate.
- `callback` (Required): A callback function that will be called with an Error object as the first parameter if there is an error.

Example

```
var sap = require('sap-access');
var cache = sap.getCache();
cache.remove('key', function(err) {
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
});
```

3.12.8.4 Using the quota service

The sap-access module lets you access the SAP API Management quota service from your Node.js code.

3.12.8.4.1 Methods

apply

Modifies the settings on a Quota object. Use this method to increment or decrement the quota, change time intervals, and make other configurations.

Usage

```
var sap = require('sap-access');
var quota = sap.getQuota();
quota.apply({parameters}, callback);
```

Example

```
var sap = require('sap-access');
var quota = sap.getQuota();

// Apply a quota of 100 requests per hour
quota.apply({
 identifier: 'Foo',
 timeUnit: 'hour',
 allow: 100
}, quotaResult);

      function quotaResult(err, r) {
       if (err) { console.error('Quota failed'); }
      }
```

Parameters

The apply() method takes two parameters, an object and a function:

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4. The first parameter is a JSON object with these fields:

   - identifier (string, required): A unique identifier of the quota bucket. In practice it might be an application ID, IP address, or username.
   - timeUnit (string, required): How long the quota bucket will accumulate until it is reset. Valid values are "minute," "hour," "day," "week," and "month."
   - allow (number, required): The maximum value for the quota bucket. This value will be combined with the current value to return whether the quota has succeeded.
   - interval (number, optional): Combined with the "timeUnit" to determine how long before the quota is reset. The default is 1. Set to a larger value to allow quotas such as "two hours," "three weeks," and so on.
   - weight (number, optional): The value to increment the quota by. Default is 1.

5. The second argument is a callback function with these two arguments:
   - The first argument is an Error object if the quota cannot be incremented, or undefined if the operation succeeded.
   - The second is an object that contains the following fields:
     - used (number): The current value of the quota bucket.
     - allowed (number): The maximum value of the quota bucket before the quota is considered to be exceeded. The same value was passed as "allow" in the request object.
     - isAllowed (boolean): If there is room left in the quota -- true as long as "used" is less than or equal to "allowed."
     - expiryTime (long): The timestamp, in milliseconds since 1970 format, when the quota bucket will be reset.
     - timestamp (long): The timestamp at which the quota was updated.

Example

```
< var sap = require('sap-access');
var quota = sap.getQuota();

// Apply a quota of 100 requests per hour
quota.apply({
  identifier: 'Foo',
  timeUnit: 'hour',
  allow: 100
}, quotaResult);

// Apply a quota of 500 requests per five minutes
quota.apply({
  identifier: 'Bar',
  timeUnit: 'minute',
  interval: 5,
  allow: 500
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
}, quotaResult);

// Increment the quota by a value of 10
quota.apply({
  identifier: 'Foo',
  timeUnit: 'hour',
  allow: 100,
  weight: 10
}, quotaResult);

function quotaResult(err, r) {
  if (err) { console.error('Quota failed'); }
```

}Put your example here>

reset

To reset the quota to zero, call quota.reset(). This method takes two parameters:

- A JSON object with these fields:
  - identifier (string, required): A unique identifier of the quota bucket. In practice it might be an application ID, IP address, or username.
  - timeUnit (string, required): How long the quota bucket will accumulate until if it is reset. Valid values are "minute," "hour," "day," "week," and "month."
  - interval (number, optional): Combined with the "timeUnit" to determine how long before the quota is reset. The default is 1. Set to a larger value to allow reset times such as "two hours," "three weeks," and so on.
- A callback function:
  - The callback takes an Error object as the first parameter if the reset fails.

3.12.8.5 Using the secure store service

The sap-access module lets you store sensitive data, such as passwords for backend services, in an encrypted format on SAP API Management.

3.12.8.6 Running in local mode vs deployed mode

To support local development and testing, the SAP API Management-access module works in a "local mode" with no dependencies on SAP API Management; however, when the module is used with an API proxy that is deployed to SAP API Management, the "local" functionality is replaced by native SAP API Management functionality. For example, the full compliment of flow variables are accessible in deployed mode, while only a small subset are available when you run the Node.js application locally. For a list of these local-mode variables, see "Running in local mode", below.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Determining the mode in which the module is running

To determine which mode you are running SAP API Management-access in:

```
var access = require('sap-access')
console.log('The deployment mode is ' + access.getMode());
```

The return value of getMode() tells you whether or not the Node.js application is deployed to SAP API Management or is running in standalone mode. The method returns one of these two string results:

- `SAP API Management` - The Node.js application is running on SAP API Management and all functionality is supported.
- `standalone` - The Node.js application is running outside the SAP API Management environment, and the default functionality described at the top of the document takes effect.

Running in deployed mode

When deployed to SAP API Management, variables set by policies are visible to SAP API Management-access, and variables added or modified by this module's methods are visible to subsequent policies in the proxy flow.

You can find the link of supported variables in the Variables Reference. These variables, and any that you create with your own names, are visible toSAP API Management-access. Note that some variables are read-only. They are identified in the Variables Reference.

Running in local mode

In "local mode," you are running your Node.js code outside the context of SAP API Management. In this mode, most of the the pre-defined flow variables are not accessible within your Node.js code. This table shows the small subset of flow variables that are available. These variables are supported here in order to support local development and testing of Node.js applications for SAP API Management.

Variable	Read-Only	Type	Notes
client.received.start.time	Yes	String	Time at which the request was received
client.received.end.time	Yes	String	Time at which the request was received
client.received.start.timestamp	Yes	Integer	Time at which the request was received
client.received.end.timestamp	Yes	Integer	Time at which the request was received

Again, on the SAP API Management platform, a much larger set of pre-defined variables is supported. Refer to the Variables Reference for the complete list.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.12.9 Debugging Node.js proxies

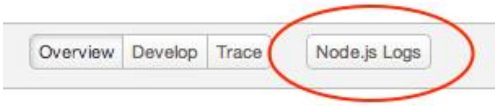
It's a good practice to make sure any Node.js code you add to a proxy works before you deploy it to SAP API Management. This topic discusses ways to debug proxies that include Node.js applications after they are deployed.

3.12.9.1 Viewing Node.js logs

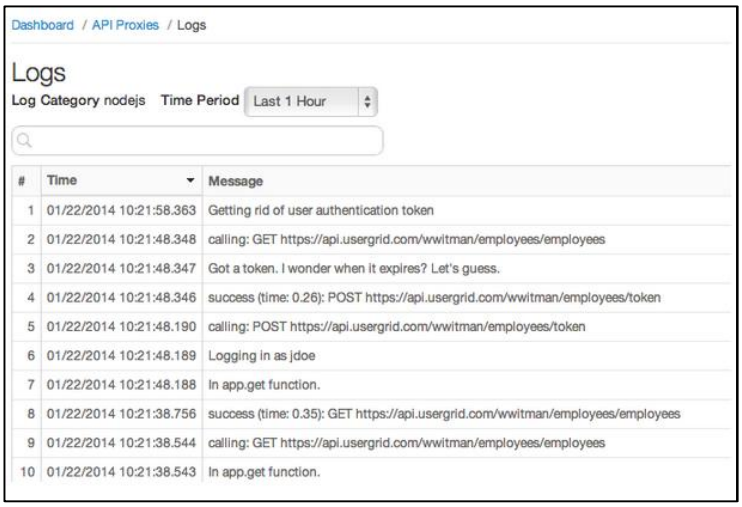
To view log information about your Node.js application:

1. In the main API proxies page, choose on the proxy you wish to view.
2. In the summary page of the selected proxy, choose Node.js Logs on the right-hand side of the tool bar.

> **Note**
>
> The Node.js Logs button does not appear for non-Node.js API proxies.



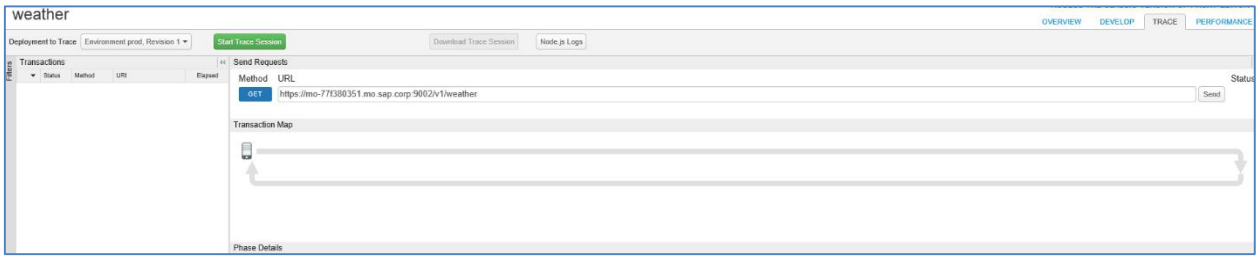
In the Logs page, you can select a time range of logs to view, as shown below. The logs record HTTP method calls, success or failure of calls, console.log messages, and so on. Enter a search string in the search field to display all log entries that contain the string.



CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.12.9.2 Using the trace tool

The trace tool is useful for general proxy debugging. For detailed information on using the trace tool, see Using the Trace tool.



3.12.9.3 Printing console output

You can embed console.log statements in your Node.js code and view the output in the trace tool. For example, the following statement prints the username of the user who is logging in to access the proxy:

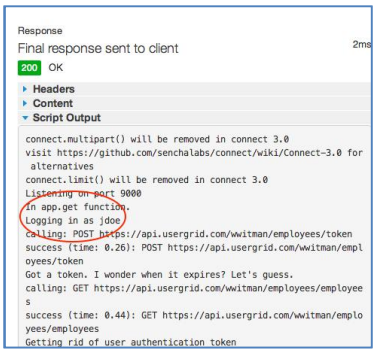
```
console.log('Logging in as %s', config.username);
```

Code: server.js

```
114         });
115     });
116 }
117
118 // We need this for UserGrid authentication
119
120 function logIn(req, res, next) {
121
122   console.log('Logging in as %s', config.username);
123
124     ug.login(config.username, config.password, function(err) {
125         if (err) {
126           console.log('Login failed: %s', JSON.stringify(err));
127           res.jsonp(500, {error: err});
128           return;
129         }
130
131         loggedIn = new usergrid.client({
132             'orgName' :  config.organization,
133             'appName' :  config.application,
134             'authType' : usergrid.AUTH_APP_USER,
135             'token':     ug.token,
136             logging:     config.logging
137         });
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

You can view the output of this console.log message in the trace tool. Simply call your API in the trace tool and open the Script Output panel of the final response, as shown below.



3.12.10 Understanding Support for Node.js Modules

Which standard Node.js modules are supported?

Use the following table to determine which standard Node.js modules are either supported, not supported, or partially supported in SAP API Management.

`assert`	Supported	
`buffer`	Supported	
`child_process`	Restricted	An exception will be thrown if an attempt is made to spawn a sub-process. However, "fork" is supported for spawning sub-scripts.
`cluster`	Disabled	The method cluster.isMaster always returns true, and other methods are not implemented. See Why the cluster module is disabled.
`crypto`	Partial support	Random and HMAC functionality is supported. The remainder is unimplemented.
`dns`	Partial support	dns.lookup is supported. All else is not implemented.
`domain`	Supported	
`dgram`	Restricted	Node.js applications in the SAP API Management environment will not be able to access services on the Internet via UDP due to our network architecture.
`events`	Supported	
`fs`	Restricted	Filesystem access is restricted to the directory where the script was launched: the /resources/nodedirectory. Node.js scripts may read and write files within this directory, for

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

		instance as a temporary scratch area, but there are no guarantees as to how long the files will persist.
`http`	Supported	The virtual host and path for incoming requests is specified in the API Proxy, not by the HTTP module. See "Understanding support for the http and https modules" for more information.
`https`	Supported	Creating an "https" server behaves identically to an "http" server. See "Understanding support for the http and https modules" for more information.
`module`	Supported	
`net`	Restricted	Attempts to listen for incoming TCP connections will generate an exception.
`path`	Supported	
`module`	Supported	
`process`	Partial support	Functionality to manipulate user ID, group membership, and working directory is not supported.
`punycode`	Supported	
`querystring`	Supported	
`readline`	Disabled	There is no standard input for scripts running on SAP API Management.
`repl`	Disabled	There is no standard input for scripts running on SAP API Management.
`module`	Supported	
`STDIO`	Supported	Standard output and error are routed to a log file within the SAP API Management infrastructure. There is no standard input for scripts running on SAP API Management. However, you can pass arguments using the ScriptTarget element of TargetEndpoint. See Advanced ScriptTarget configuration for more information.
`stream`	Supported	
`string_decoder`	Supported	
`timers`	Supported	
`tls`	Supported	Transport Layer Security (TLS) parameters work basically the same way they work in regular

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

		Node.js. See Using the TLS (SSL) Node.js module on SAP API Management for details.
`tty`	Disabled	There is no standard input for scripts running on SAP API Management.
`url`	Supported	
`util`	Supported	
`vm`	Supported	
`zlib`	Supported	

3.12.10.1 Understanding support for the http and https modules

All Node.js applications running in SAP API Management must use the http or https module to listen for incoming requests. If you were to deploy a script that doesn't listen for incoming requests, it would simply execute and exit.

The listen method of the http and https modules in Node.js takes a port number as a parameter. For example:

```
svr.listen(process.env.PORT || 9000, function() {
   console.log('The server is running.');
});
```

This "port" argument is required in Node.js, but SAP API Management ignores this parameter. Instead, the API proxy in which the Node.js script runs specifies the "virtual host" that it listens on, and the Node.js application uses those same virtual hosts, just like any other SAP API Management proxy.

Every environment in SAP API Management has at least one virtual host. The virtual host defines the HTTP settings for connection with the SAP API Management organization. All API proxies in an environment share the same virtual hosts. By default, two virtual hosts are available for each environment: default and secure.

The SAP API Management tool deploynodeapp command generates an SAP API Management proxy wrapper around the Node.js application. When deployed, the Node.js application listens on the default virtual host defined for the environment.

Handling incoming requests

Like other SAP API Management applications, if the proxy application is set up to listen on the secure virtual host, then it will accept incoming requests using HTTPS.

Handling outgoing requests

In addition to receiving incoming traffic, Node.js applications inside SAP API Management may use the http and https modules to make outbound requests like any other Node.js application. These modules work just as they always do inside Node.js.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.12.10.2 Understanding support for the tls module

SAP API Management supports the Node.js tls module. This module uses OpenSSL to provide Transport Layer Security (TLS) and/or Secure Socket Layer (SSL) encrypted stream communication. You can use the tls module to create secure connections to backend services from Node.js applications running on SAP API Management.

To understand how the tls module works on SAP API Management, it's important to understand how virtual hosts are used on SAP API Management. Every environment in SAP API Management has at least one virtual host. The virtual host defines the HTTP settings for connection with the SAP API Management organization. All API proxies in an environment share the same virtual hosts. By default, two virtual hosts are available for each environment: default and secure.

Now, let's look at how SAP API Management handles TLS (SSL) communication for incoming and outgoing requests on Node.js applications:

Handling incoming requests

Depending on how virtual hosts are configured for your organization, SAP API Management provides these options:

If the API proxy is configured to listen on the default virtual host, then it accepts requests over HTTP.

Handling outgoing requests

You can make outgoing requests with the tls module the same way you would normally in Node.js. Basically, you need to add client-side keys and certificates (.pem files) to the resources/node directory and load them inside your script. For information on using the tls module and its methods, see the Node.js tls module documentation.

3.12.10.3 Why the cluster module is disabled

SAP API Management does not support the "cluster" module, which allows Node.js applications to control when additional copies of an application are started and stopped, and to communicate with them. In the parlance of the "cluster" module, all Node.js applications are considered to be the "master."

However, SAP API Management does deploy multiple instances of each application — two in most cases, although the system may be configured to run more if necessary.

That means that Node.js applications running inside SAP API Management, like all Node.js applications, must assume that they will be processing many different requests "in parallel" using a single thread of control. However, the application must not assume that it is the only copy running in the world, though it also must not assume that there is some way to communicate with the other copies.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.12.10.4 Advanced ScriptTarget Configuration

In the <TargetEndpoint> definition, the <ScriptTarget> element takes additional optional parameters besides <ResourceURL>. You can also pass command-line arguments and environment variables to a Node.js script using the <EnvironmentVariables> and <Arguments> parameters:

```
<TargetEndpoint name="default">
  <ScriptTarget>
     <ResourceURL>node://hello.js</ResourceURL>
     <EnvironmentVariables>
         <EnvironmentVariable name="NAME">VALUE</EnvironmentVariable>
     </EnvironmentVariables>
     <Arguments>
         <Argument>ARG</Argument>
     </Arguments>
  </ScriptTarget>
</TargetEndpoint>
```

3.13 Debug and Troubleshoot

3.13.1 Using a Trace Tool

Trace is a tool for troubleshooting and monitoring API proxies running on SAP API Management. Trace lets you probe the details of each step through an API proxy flow.

3.13.2 How to use Trace

Trace is simple to use. You start a trace session and then make an API call to the SAP API Management, and read the results.

1. Select API Proxies from the APIs menu.
2. Select an API proxy from the API Proxies page.
3. Be sure the API you wish to trace is deployed.
4. Choose Trace to go to the Trace tool view.
5. Use the Deployment to Trace dropdown menu to select which deployment environment and proxy revision you wish to trace.
6. Choose Start Trace Session. When the Trace session is active, the API proxy records details of each step in the processing pipeline. While the Trace session is running, messages and contextual data are captured from live traffic.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7. If you don't have any live traffic flowing through your proxy, then simply send a request to the API. You can use whatever tool you wish to send the request, such as cURL, the SAP API Management API console, Postman, or any familiar tool. Or, you can send the request directly from the Trace tool itself. Just enter the URL and choose Send.

   Note

   One Trace session can support 10 request/response transactions through the selected API proxy. If there are 2 per router, then 20 request/response transactions are supported. Trace sessions automatically stop after 10 minutes if you don't manually stop it.
8. When you've captured a sufficient number of requests, choose Stop Trace Session.
9. A list of captured request/response transactions displays in the left menu. Choose on any of the transactions to view detailed results.

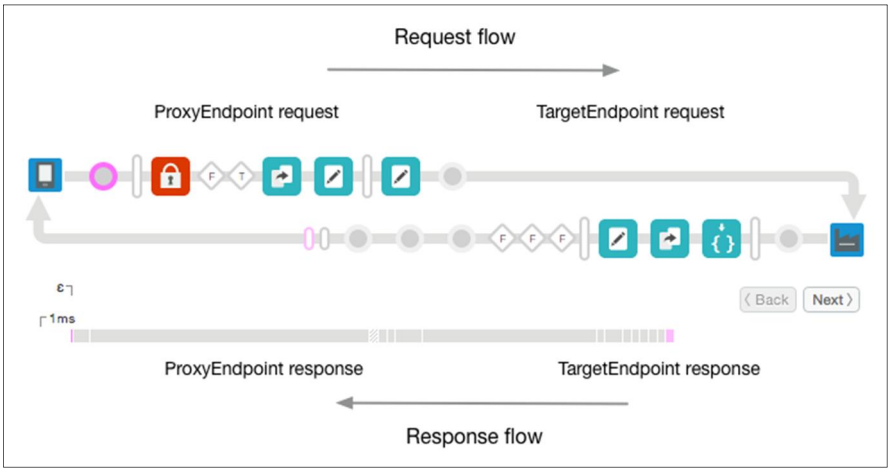
3.13.3 How to read a Trace

To read a trace, just follow the arrows: the request flow steps are shown along the top and response flow steps along the bottom of the map.

The tool's transaction map uses icons to mark each notable step that occurs during an API proxy transaction, including policy execution, conditional steps, and transitions. Hover over any icon to see summary information.

The phase details section of the tool lists information about the proxy's internal processing, including variables that were set or read, request and response headers, and much more. Choose any icon to see the phase details for that step.

Here's a sample trace tool map with the main proxy processing segments labeled:



© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.13.4 Transaction map legend

Any good map needs a legend. The following table describes the intent of the icons you will see in the transaction map. These icons mark each of the notable processing steps throughout the proxy flow.

Transaction map icons

	The client app that sends a request to the ProxyEndpoint of the API proxy.
	The circles mark transitional endpoints in the proxy flow. They are there when a request comes in from the client, when the request goes to the target, when the response comes back from the target, and when the response goes back to the client.
	The tall bars indicate the beginning of a flow segment in the API proxy flow. Flow segments are: ProxyEndpoint request, TargetEndpoint request, TargetEndpoint response, and ProxyEndpoint response. A segment includes the PreFlow, Conditional Flows, and PostFlow. See Flows Configuration for more information.
	The short bars are called "executions." You'll see these bars when Analytics actions occur in the background. Hover over these bars to see what kind of data is being stored in Analytics.
	A conditional flow that evaluates to true. For an introduction to conditional flows, See Flows Configuration for more information.
	A conditional flow that evaluates to false. For an introduction to conditional flows, See Flows Configuration for more information.
	A policy. Each type of policy has a unique icon. This one is for the AssignMessage policy. These icons let you see where policies are executed in the proper order and if they are successful or not. You can choose a policy icon to see the results of its execution and if they are expected or not. For example, you can see if the message was transformed properly or if it is being cached. Properly executing policies are clearly indicated by check-marks. In the case of an error, a red exclamation mark is displayed on the icon. Tip: Pay attention to the tooltip or the time line to see if any policy is taking longer than expected.
	The backend target called by the API proxy.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

	The time line indicates how long (in milliseconds) that the processing time took to complete. Comparing the elapsed time segments helps you isolate the policies that are taking the longest to execute that are slowing down your API calls.
	The Epsilon indicates a time-span smaller than a millisecond.
	Disabled. Appears on a policy icon when a policy is disabled. A policy can be disabled with the public API. See API proxy configuration reference.
	Error. Appears on a policy icon when the Policy Step condition evaluates to false (see Flow Variables and Conditions), or on the RaiseFault policy icon whenever a RaiseFault policy executes.
	Skipped. Appears on a policy icon when the policy was not executed because the step condition evaluated to false. See Flow Variables and Conditions for more information.

3.13.5 Understanding the phase details

The Phase Details part of the tool tells you a lot about the state of your proxy at each processing step. Here are some of the details provided in the Phase Details. Choose any icon in the trace tool to see details for the selected step, or use the Next / Back buttons to move from one step to another.

> **Note**
>
> Trace captures message content. If your message payloads contain sensitive information, then you should enable data masking. For instructions, see Data masking.

Phase Detail	Description
Proxy Endpoint	Indicates which ProxyEndpoint flow was selected for execution. An API proxy can have multiple named proxy endpoints.
Request Headers	Lists the HTTP request headers.
Request Content	Shows the HTTP request body.
Properties	Properties represent the internal state of the API proxy. These are not shown by default.
Variables Read	Lists the flow variables that were read by a policy. See also Introduction to Flow Variables.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Phase Detail	Description
Variables Read and Assigned	Lists the flow variables that were read and assigned a value by a policy.
Target Endpoint	Indicates which TargetEndpoint was selected for execution.
Response Headers	Lists the HTTP response headers.
Response Content	Shows the HTTP response body.
PostClientFlow	Shows information about the PostClientFlow, which executes after the request, is returned to the requesting client app. Only MessageLogging policies can be attached to the PostClientFlow. The PostClientFlow is currently used primarily for measuring the time interval between the start and end timestamps for the response message.

3.13.6 Debugging with Trace

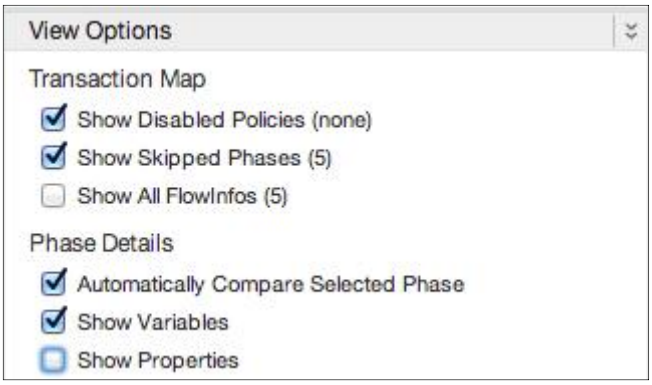
Trace lets you see a lot of internal details about an API proxy. For example:

- You can see at a glance which policies are executing correctly or failing.
- Let's say you noticed through one of the Analytics dashboards that one of your APIs is experiencing an unusual decrease in performance. Now, you can use Trace to help identify where the bottleneck is occurring. Trace gives the time, in milliseconds, which it takes for each processing step to complete. If you find one step is taking too long, you can take corrective action.
- By looking at the phase details, you can check headers that are being sent to the backend, view variables set by policies, and so on.
- By verifying the base path, you can ensure that a policy is routing the message to correct server.

3.13.7 Selecting View Options

- Choose the view options for the trace session.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.



Option	Description
Show Disabled Policies	Show any disabled policies. A policy can be disabled with the public API. See API proxy configuration reference.
Show Skipped Phases	Show any phases that were skipped. A skipped phase occurs when policy was not executed because the step condition evaluated to false. See Flow Variables and Conditions for more information.
Show all FlowInfos	Represent transitions within a flow segment.
Automatically Compare Selected Phase	Compares the selected phase to the previous one. Turn this off to see only the selected phase.
Show Variables	Show or hide variables that were read and/or assigned a value.
Show Properties	Properties represent the internal state of the API proxy. (Hidden by default.)

3.13.8 Downloading trace results

You can download an XML file of the trace results for viewing offline. The file shows the complete details of the listening session including the contents of all headers, variables and policies.

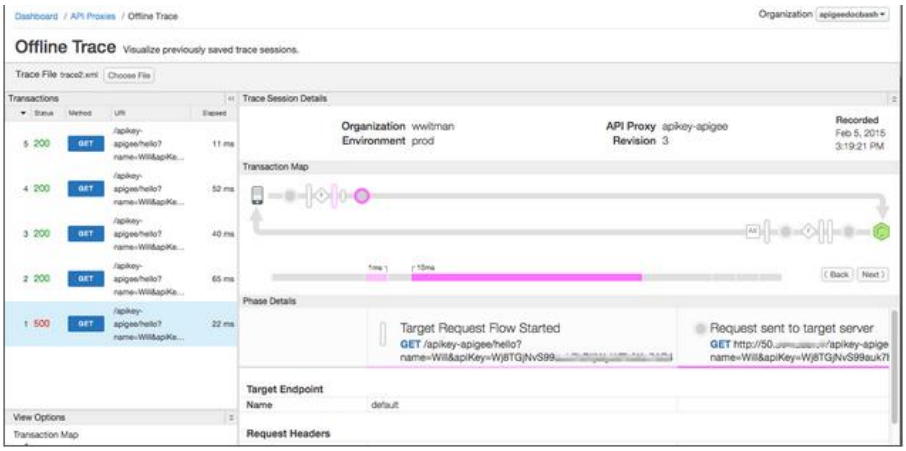
To download raw trace output, choose Download Trace Session.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.13.9 Using Offline Trace Tool

3.13.9.1 What is the Offline Trace tool?

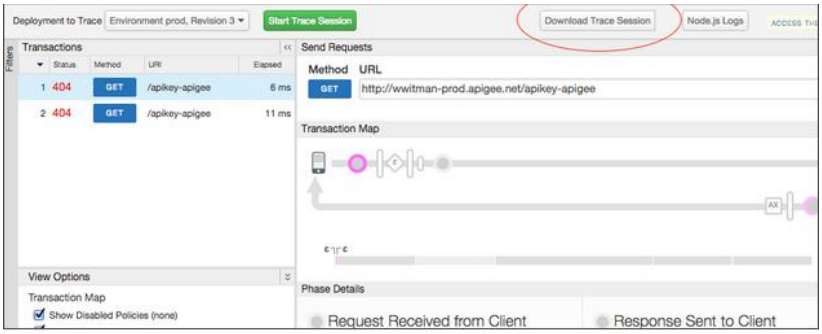
The Offline Trace tool lets you view and analyze trace sessions that were previously saved. A saved trace session is essentially a "recording" of a trace session, and can be useful for cases where troubleshooting and further analysis is required. The UI for the Offline Trace tool is similar to the "live" Trace tool. To learn about the Trace Tool UI and on saving trace sessions, see Using a Trace tool.



3.13.9.2 Downloading a trace session

To download a trace session:

1. Open the Trace Tool, as described in Using the Trace tool.
2. Perform a trace.
3. When the trace is completed, select Download Trace Session.

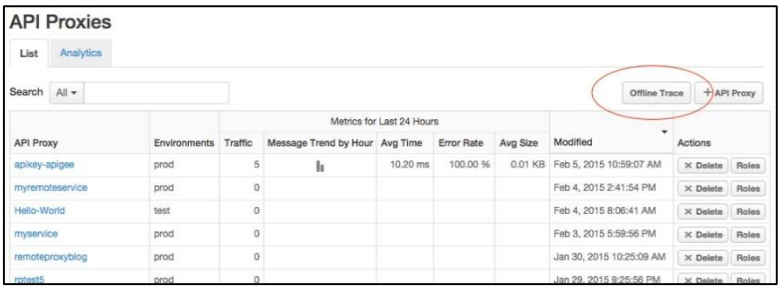


The downloaded trace is stored in XML format.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

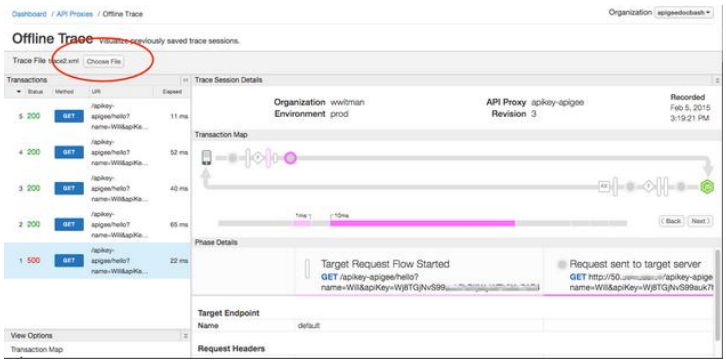
3.13.9.3 Opening the Offline Trace tool

1. Select APIs > API Proxies from the main SAP API Management menu.
2. On the API Proxies page, choose Offline Trace.



3.13.9.4 Loading a trace file

In the Offline Trace tool, choose Choose File and select a downloaded trace file from your system.



3.13.10 Troubleshooting

3.13.10.1 Refining message capture using filters

To enable root-case analysis, filters enable you to target specific calls that may be causing problems. For example, you may need to zero in on requests that have specific content or requests coming from specific partners or apps.

- HTTP headers - limits the trace to only calls that contain a specific header. This is a good way of helping you troubleshoot issues. You can send a header to your app developer and ask them to include it in the call that is

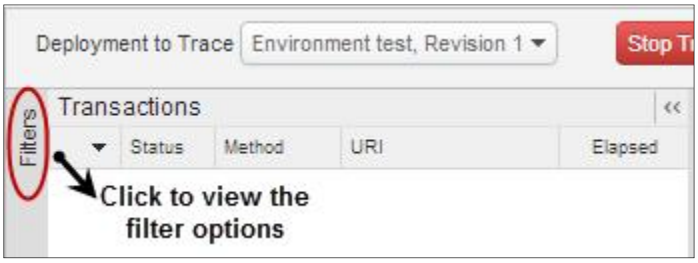
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

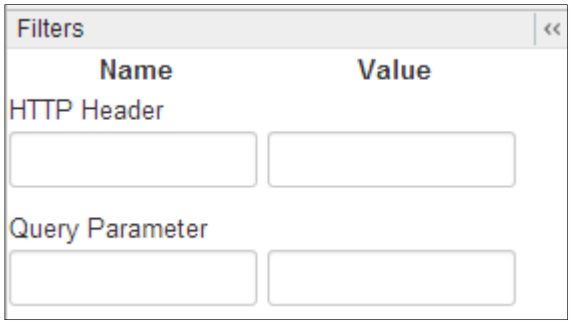
causing issues. Then SAP API Management will only record calls with that specific header so you can examine the results.

- Query parameters - only calls with a specific value of a parameter will be recorded.

To add a filter:

1. Open the Filter section of the Trace page.





2. Enter a name and value into either the Header or Query fields. Blank fields are not evaluated.

   You can add more filters by choosing the Add Filter button. You can delete filters by choosing the (X) button next to the filter.

3.13.10.2 HTTP Response Codes

Because the management API is RESTful, response messages can be interpreted as a combination of an HTTP response code and an SAP API Management -specific error message.

For example, if you try to create a company entity with the same name as an existing company, the response is:

```
404
{
  "code": "messaging.config.beans.ApplicationDoesNotExist",
  "message": "APIProxy named WeatherApi does not exist in organization mycompany",
  "contexts": []
}
```

An explanation of important HTTP response codes follows:

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- HTTP 2xx: Any response code in the 200 range means 'success'. In some cases only a 204 HTTP response code is issued when an operation succeeds. 204 means that the response is submitted with no content, usually because a DELETE operation succeeded.
- HTTP 401: This is an authorization failure. It means that the credentials that you used to make a request map to a user who lacks permission to perform the operation. Verify the roles for the account you are using.
- HTTP 403: This is an authorization failure. It means that the username and password combination you are using is not valid for the organization you specified. To test your credentials, login to management UI using <host: port>/login. Once logged in check your account settings to see the organization name. In some situations, you may belong to multiple organizations. Make sure that you are using the correct credentials for you organization. Check your spelling.
- HTTP 404: The object wasn't found. This is usually happens because of misspelling in your request URL, but it may also happen if you try to update an object that doesn't exist (for example, the wrong revision of an API).
- HTTP 405: This means 'Method not allowed'. This response code simply means that you, for example, used the GET verb for an API call that requires the POST verb.
- HTTP 415: Unsupported media type. This error usually occurs on POST or PUT requests when the Content-typeHTTP header is set to the wrong value. For example, imagine that you POST the following to an API that only supports JSON:

  ```
  $ curl -X POST -H "Content-type:text/xml" -d '<SomeXML>'
  https://api.company.com/v1/json_service
  ```

The result would be an HTTP 415 error.

For GET requests, remember to use the Accept header instead of the Content-type header

.

- HTTP 500: This is an internal server error. It means that you should contact the administrator to resolve the issue.

3.13.10.3 Common Client Issues

Using cURL on Windows

All of the example commands use Linux/Unix command line syntax for quoting arguments. A single quote can be used as a wrapper for double-quote characters inside a pair of single quotes.

If you are using cURL from a Windows command prompt, Windows does not provide such multi-layered quoting. Under Windows, the outermost quotes must be the double-quote character and quotes within that string must doubled, two double-quote characters, where the following examples have one.

curl and HTTPS

The management API requires SSL (HTTPS). If you encounter SSL Certificate problems in response to your cURL commands, you can:

- Update your certs file.
- Use a command line option to specify an alternate certs file.

Protocol https not supported or disabled in libcurl

Usually seen on machines running Windows, This means that you downloaded and installed a version of curl that did not include the libcurl package. Download a different package from http://curl.haxx.se/download.html, and make sure it includes libcurl for SSL.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

CLASSIFICATION_FAILURE

You may see this error when you submit a request to your API proxy after you deploy it to API Platform. This error means that API Platform, while attempting to route the request to the appropriate API proxy, cannot find an API proxy that matches the request URI path. This usually happens because your request URL does not match the base path for any API proxies currently deployed in the environment. (Note that API Platform validates that all base paths are unique when a new API proxy is imported or generated.)

- Verify the URI for your API by logging in to the SAP Management UI. Select the link to your API. At the top of the page, the URI for you API including the base path displays.
- Make sure your API proxy is deployed. Your API proxy may be imported to SAP API Management, but may not be deployed to an environment. API proxies can only be invoked after they have been successfully deployed to an environment. (Note that the deploy tool used in the quick start both imports and deploys your API proxy to the environment you specify.)

3.13.10.4 Deploy Tool Errors

`python: can't open file 'tools/deploy.py': [Errno 2] No such file or directory`

The path you have provided to deploy.py is incorrect.

If you see the following result on import, it indicates that you are running the deploy tool from the wrong directory.

```
Import failed to /v1/organizations/myorg/apis?action=import&name=weatherapi with
status 500:
{
  "code" : "messaging.config.beans.ImportFailed",
  "message" : "Failed to import the bundle : java.util.zip.ZipException: ZIP file
must have at least one entry",
  "contexts" : [ ]
}
```

To fix this issue, run deploy.py from the base directory of the API Platform samples distribution. In the deploy tool command, specify the directory that contains the `./apiproxy directory`-- in this package that value is `simpleProxy`.

3.13.11 View API History

SAP API Management provides a "history" feature that can be helpful in troubleshooting problems as well as managing API usage. The history feature enables you to view the operations (create, update, read, delete, deploy, undeploy) that have been performed on your APIs and API products over time. You can see who performed each operation, when the operation was performed, the request URI, and the response code. For update operations, you can also view the request body that was as passed in the API call.

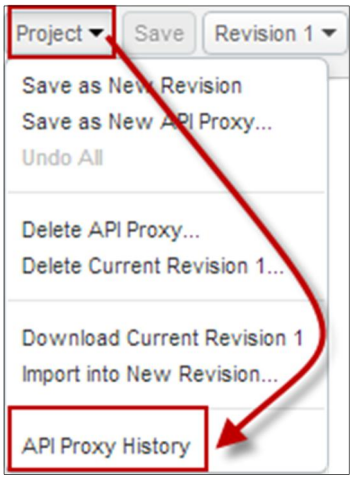
CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.13.11.1 Retrieving API History

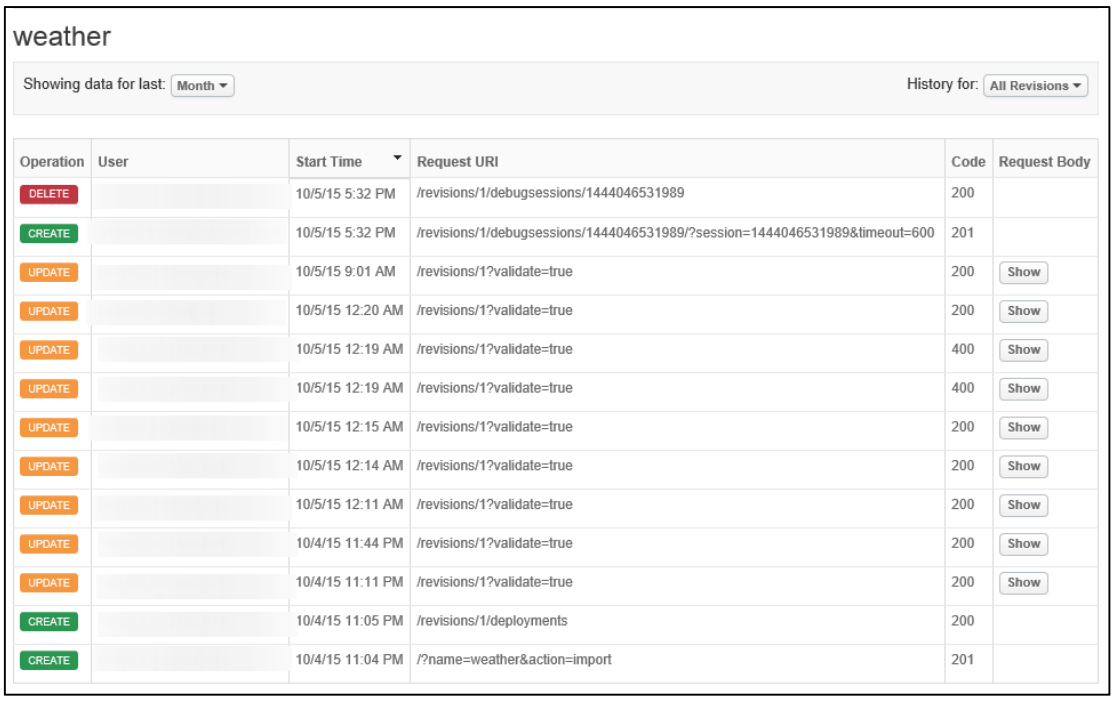
To view the history of an API:

1. Select the API from the summary list on the API Proxies page.
2. On the API detail page, choose Project, and choose API Proxy History.

   This displays the history for the API.



   The history of the selected API is rendered.



3. You can change the date range for the history to the last day, week, 2 months, 3 months, or year, by selecting from the menu next to "Showing data for last".

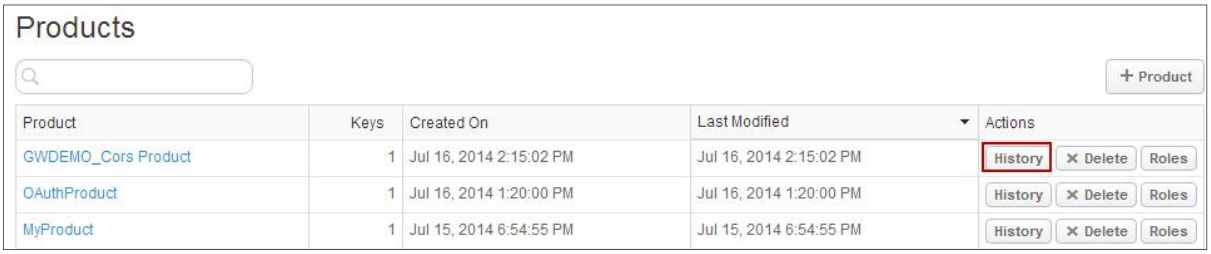
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4. You can view the request body for update operations by choosing Show in the Request Body column for the operation.

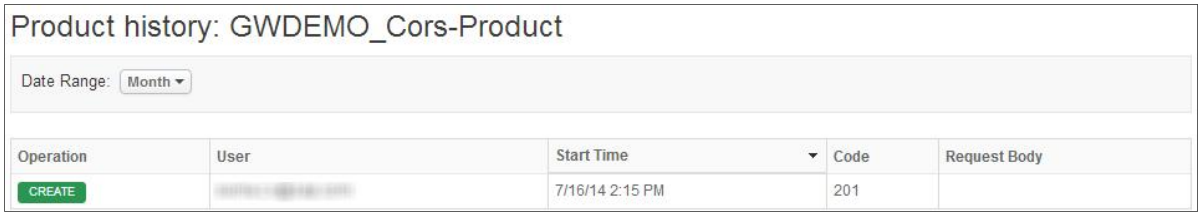
3.13.11.2 Retrieving API product history

To view the history of an API product:

1. Select History in the Actions column for the product on the Products page.



This history for the API product is displayed.



2. You can change the date range for the history to the last day, week, 2 months, 3 months, or year, by selecting from the menu next to Date Range.
3. You can view the request body for update operations by choosing Show in the Request Body column for the operation.

3.14 Environment Configurations

3.14.1 Creating and Editing an Environment Cache

When you don't want to use the included shared cache, you can create and configure your own cache. You can use the cache you create beneath caching policies, rather than using the the shared cache.

> **Note**
>
> Under ordinary circumstances, you don't need to configure your own cache. Only configure a cache when you need to customize the cache settings and optimize performance.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.14.1.1 Defining a cache resource

You can create multiple cache resources in each environment. When you're configuring a caching policy, you specify whether the policy should use the included shared cache or a cache you created.

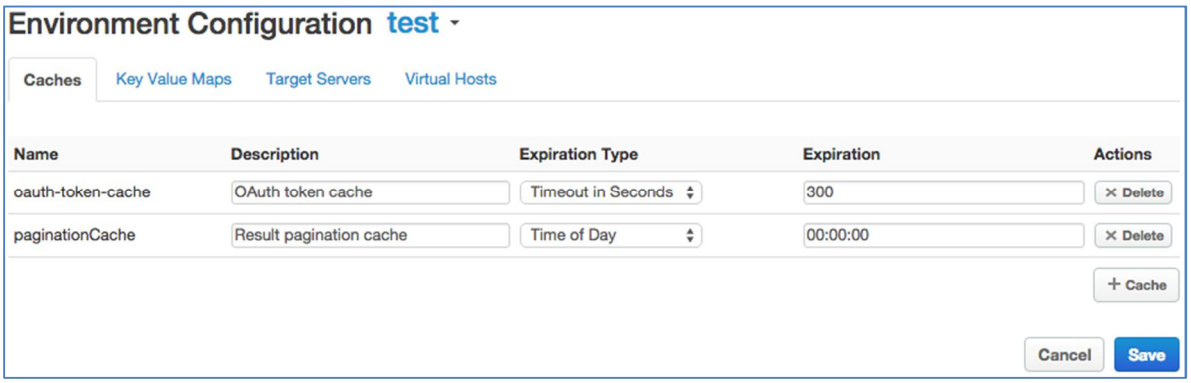
For data segregation, the scope of a cache is limited to the environment in which you created the cache. (For example, API proxies running in a 'test' environment cannot access data in a cache running in 'prod'.) Once you create a cache, policies can populate it with any serializable data.

3.14.1.2 Creating and editing caches

These steps describe how to create or edit a cache using the management console.

1. In the management UI, choose the APIs menu, then choose Environment Configuration.
2. In the Environment dropdown, select the environment for which you want to configure caches, such as test or prod.
3. Go to the Caches tab.
4. Choose Edit.
5. Under Caches, at the right side, choose the +Cache button to add a new cache.

   The button displays the New Cache dialog.



6. In the New Cache form, enter property values for the new cache. The following table describes the settings.

Property Name	Default Value	Description	Notes
Name	mycache	The name of the cache. Must be unique in the environment. Reference this cache from policies that interact with the cache resource.	

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Property Name	Default Value	Description	Notes
Description	N/A	An optional description of the cache resource.	
Expiration Type	Timeout in Seconds	Sets how cache entries will be expired. Entry time to live can be a specified number of seconds after creation, a specified time of day each day, or a specified date.	Populate Cache policyand Response Cache policy both override these expiration settings with their own for cache entries they create.
Expiration	Timeout: 300 (seconds) Time of day: 12:00:00 Date: current date (dd-MM-yyyy)	The configuration setting (either integer or dateTime) for the selection you made in the Expiration Type dropdown. Enter time of day in the format HH:mm:ss, where HH represents the hour on a 24-hour clock. For example, 14:30:00 for 2:30 in the afternoon. For the time of day, the default locale and time zone will vary depending on where the code is running (which isn't knowable at configuration time).	

7. Choose Save.

3.14.2 Creating and Editing an Environment Key/Value Maps

There are often situations where you want to store key/value pairs of data for longer term and access that data at runtime with your API proxies.

For example, you could store the latitude and longitude of a location of a conference (using key names like conLAT and conLON), retrieve those keys in an API proxy call, and pass the latitude and longitude data into the message body of the request or response (or to a mapping API). And if the location of the conference happened to change the following year, all you would have to do is update the values without having to touch your API proxy logic, which gets the data using the same key names.

You store one or more key/value pairs of data in a grouping called a 'map', or 'key/value map'.

Caution:

Key/value maps aren't designed to store sensitive data such as usernames and passwords.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.14.3 About scope

There may be different scopes at which you want to store key/value data. For example, if only one API proxy requires data in a key/value map, you can create the key/value map at the API proxy scope, where only that API proxy can access the data. Or you may want all API proxies in your test environment to have access to a key/value map, in which case you'd create a key/value map at the environment scope.

Note:

When creating and managing key/value maps in the management UI, which this topic covers, you're creating them at the environment scope.

or information about how to create and manage key/value maps at other scopes, such as organization or proxy scopes (as well as the environment scope), see the following topics:

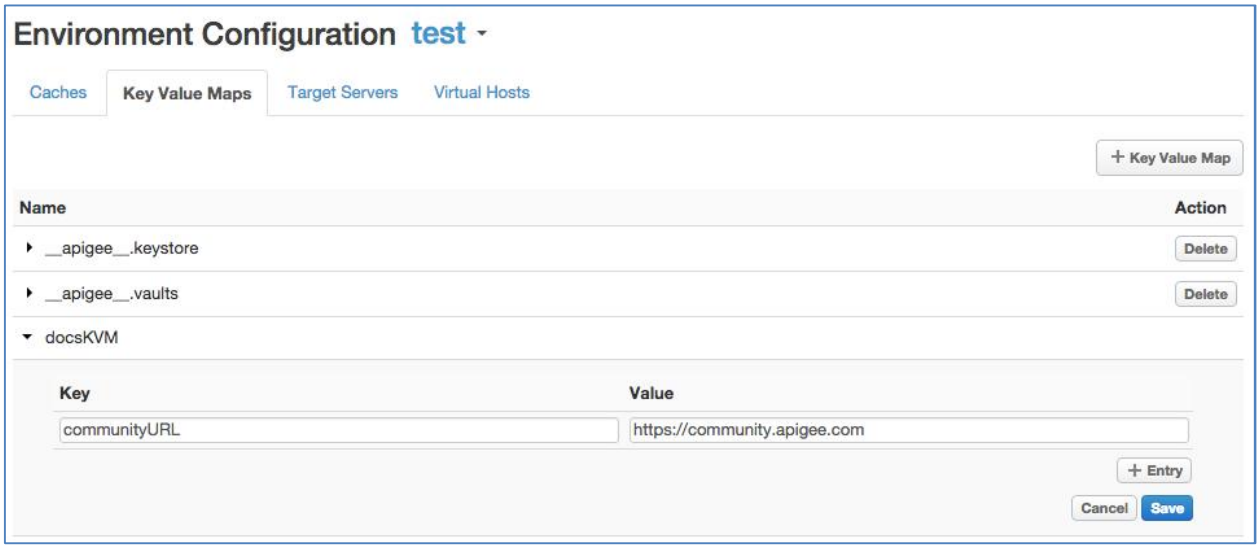
Management API: Persist Data Using KeyValueMap.

Policy: Configuring the KeyValueMapOperation Policy. The policy is also how you access key/value data at runtime.

Creating and editing environment key/value maps

These steps describe how to create or edit a key/value map in the management UI. Key/value maps created this way are at the environment scope. For example, if you create a key/value map in the 'test' environment, API proxies running in the 'prod' environment won't have access to the key/value map.

1. In the management UI, select APIs > Environment Configuration.
2. Select the environment for which you want to configure key/value maps, such as test or prod.
3. Go to the Key Value Maps tab.
4. Click +Key Value Map to create an new one, or expand an existing key/value map and click Edit.
5. Click the +Entry button to add a new key/value pair.



6. Enter key/value properties as described in the following table.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Property Name	Default Value	Description
Key	N/A	The key name that you'll use to access the data value in your API proxies. In organizations with Core Persistence Services (see below), key names cannot be larger than 2 KB.
Value	N/A	The value of the key. Enter any combination of numbers, letters, or special characters. There is no size limitation.

7. Choose Save.

To view or modify maps with more than 150 keys, the UI directs you to use the SAP API Management's API.

3.14.4 Load Balancing Across Backend Servers

SAP API Platform enhances the availability of your API by providing built-in support for load balancing and failover across multiple backend server instances.

TargetServer configurations decouple concrete endpoint URLs from TargetEndpoint configurations. Each TargetServer is referenced by name in a TargetEndpoint HTTPConnection. Instead of defining concrete URL in the configuration, you can configure one or more named TargetServers.

A TargetServer definition consists of a name, a host and a port, with an additional element to indicate whether the TargetServer is enabled or disabled.

Sample TargetServer Configuration:

```
<TargetServer  name="target1">
  <Host>1.mybackendservice.com</Host>
  <Port>80</Port>
  <IsEnabled>true</IsEnabled>
</TargetServer >
```

TargetServer Configuration Elements

Name	Description	Default	Required?
name	The name of the TargetServer configuration. The name must be unique within the environment.	N/A	Yes

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Default	Required?
Host	The host URL of the backend service.	N/A	Yes
Port	The port on which the backend service is listening	N/A	Yes
IsEnabled	A boolean that indicates whether the TargetServer configuration is enabled or disabled. This enables you to take TargetServers out of rotation without modifying the API proxy configuration. A common usage would be to write an app or script that enables or disables TargetServers automatically based on expected capacity requirements, maintenance schedules, etc.	true	Yes

To create a TargetServer in an environment:

```
$ curl -H "Content-Type:text/xml" -X POST -d \
'<TargetServer name="target1">
   <Host>1.mybackendservice.com</Host>
   <Port>80</Port>
   <IsEnabled>true</IsEnabled>
 </TargetServer>' \
-u myname:mypass http://<host:port>/v1/o/{org_name}/environments/test/targetservers
```

Sample Response:

```
{
  "host" : "1.mybackendservice.com",
  "isEnabled" : true,
  "name" : "target1",
  "port" : 80
}
```

After you create the first TargetServer, then create a second TargetServer. By defining two TargetServers, you provide two URLs that a TargetEndpoint can use for loadbalancing:

```
$ curl -H "Content-type:text/xml" -X POST -d \
'<TargetServer  name="target2">
  <Host>2.mybackendservice.com</Host>
  <Port>80</Port>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <IsEnabled>true</IsEnabled>
  </TargetServer >' \
  -u myname:mypass http://<host:port>/v1/o/{org_name}/environments/test/targetservers
```

Sample Response:

```
  {
    "host" : "2.mybackendservice.com",
    "isEnabled" : true,
    "name" : "target2",
    "port" : 80
  }
```

Retrieve a list of TargetServers in an environment:

```
  $ curl -u myname:mypass
  http://<host:port>/v1/o/{org_name}/environments/test/targetservers
```

Sample response:

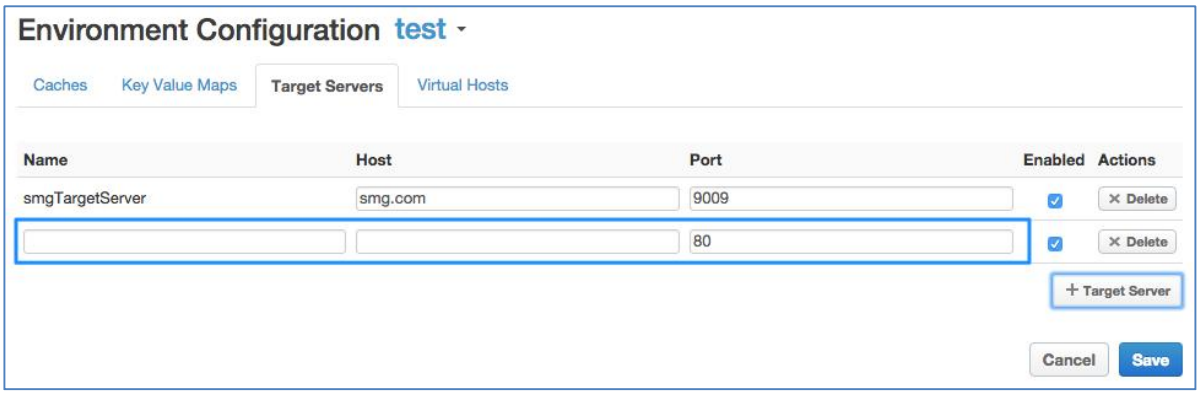
```
  [ "target2", "target1" ]
```

There are now two TargetServers available for use by API proxies deployed in the test environment. To load balance traffic across these TargetServers, you configure the HTTP connection in an API proxy's target endpoint to use the TargetServers.

There is no limit on the number of TargetServers that you can set up in an environment or reference from a TargetEndpoint's HTTPConnection.

3.14.5 Managing target servers with the UI

You can use the SAP API Management Edge UI to create and manage target servers.



1. Select APIs > Environment Configuration in the management UI.
2. Select the desired environment, such as test or prod.
3. Go to the Target Servers tab.
4. Click Edit.
5. Click + Target Server.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6. Enter a name, host, and port.
7. Be sure Enabled is selected.
8. Click Save.

For example:

- Name: target1
- Host: 1.mybackendservice.com
- Port: 80
- Enabled: Selected

Repeat these steps to add and enable more target servers.

3.14.6 Managing target servers with APIs

You can use Edge management APIs to create, delete, update, get, and list target servers. For more information, see TargetServers.

```
$ curl -H "Content-Type:text/xml" -X POST -d \
'<TargetServer name="target1">
   <Host>1.mybackendservice.com</Host>
   <Port>80</Port>
   <IsEnabled>true</IsEnabled>
 </TargetServer>' \
-u email:password
https://<managment_server_ip:8080>/v1/o/{org_name}/environments/test/targetservers
```

Sample Response:

```
{
  "host" : "1.mybackendservice.com",
  "isEnabled" : true,
  "name" : "target1",
  "port" : 80
}
```

After you create the first TargetServer, then create a second TargetServer. By defining two TargetServers, you provide two URLs that a TargetEndpoint can use for loadbalancing:

```
$ curl -H "Content-type:text/xml" -X POST -d \
'<TargetServer  name="target2">
  <Host>2.mybackendservice.com</Host>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  <Port>80</Port>
  <IsEnabled>true</IsEnabled>
</TargetServer >' \
-u email:password
https://<managment_server_ip:8080>/v1/o/{org_name}/environments/test/targetservers
```

Sample Response:

```
{
  "host" : "2.mybackendservice.com",
  "isEnabled" : true,
  "name" : "target2",
  "port" : 80
}
```

Retrieve a list of TargetServers in an environment:

```
$ curl -u email:password
https://<managment_server_ip:8080>/v1/o/{org_name}/environments/test/targetservers
```

Sample response:

```
[ "target2", "target1" ]
```

There are now two TargetServers available for use by API proxies deployed in the test environment. To load balance traffic across these TargetServers, you configure the HTTP connection in an API proxy's target endpoint to use the TargetServers.

There is no limit on the number of TargetServers that you can set up in an environment or reference from a TargetEndpoint

3.14.7 Configuring a TargetEndpoint to load balance across named TargetServers

Now that you have two TargetServers available, you can modify the TargetEndpoint HTTP connection setting to reference those two TargetServers by name.

```
<TargetEndpoint name="default">
  <HTTPTargetConnection>
    <LoadBalancer>
      <Server name="target1" />
      <Server name="target2" />
    </LoadBalancer>
    <Path>/test</Path>
  </HTTPTargetConnection>
</TargetEndpoint>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The configuration above is the most basic load balancing configuration possible. The load balancer supports three load balancing algorithms, Round Robin, Weighted, and Least Connection. Round Robin is the default algorithm. Since no algorithm is specified in the configuration above, outbound requests from the API proxy to the backend servers will alternate, one for one, between target1 and target 2.

> **Note**
>
> Retries are triggered by I/O exceptions and HTTP time outs. Retries are not triggered by HTTP error codes (4xx, 5xx).

3.14.7.1 Setting Load Balancer Options

You can tune availability by using options for load balancing and failover at the load balancer and TargetServer level.

Available options for LoadBalancers are:

3.14.7.1.1 Algorithm

Sets the algorithm used by `LoadBalancer`. The available algorithms are `RoundRobin, Weighted`, and `MaximumFailures`, each of which is documented below.

Round robin

The default algorithm,RoundRobin forwards a request to each TargetServer in the order in which the servers are listed in the target endpoint HTTP connection.

For example:

```
<TargetEndpoint name="default">
  <HTTPTargetConnection>
      <LoadBalancer>
        <Algorithm>RoundRobin</Algorithm>
        <Server name="target1" />
        <Server name="target2" />
      </LoadBalancer>
      <Path>/test</Path>
  </HTTPTargetConnection>
</TargetEndpoint>
```

Weighted

The weighted load balancing algorithm enables you to configure proportional traffic loads for your TargetServers. The weighted LoadBalancer distributes request to your TargetServers in direct proportion to each TargetServer's weight. Therefore, the weighted algorithm requires you to set a weight attribute for each TargetServer. For example:

```
<TargetEndpoint name="default">
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <HTTPTargetConnection>
      <LoadBalancer>
        <Algorithm>Weighted</Algorithm>
        <Server name="target1">
          <Weight>1</Weight>
        </Server>
        <Server name="target2">
          <Weight>2</Weight>
        </Server>
      </LoadBalancer>
      <Path>/test</Path>
    </HTTPTargetConnection>
  </TargetEndpoint>
```

In this example, 2 requests will be routed to target2 for every 1 request routed to target1.

Least Connection

LoadBalancers configured to use the least connection algorithm route outbound requests to the TargetServer with fewest open HTTP connections.

```
  <TargetEndpoint name="default">
    <HTTPTargetConnection>
        <LoadBalancer>
          <Algorithm>LeastConnection</Algorithm>
          <Server name="target1" />
          <Server name="target2" />
        </LoadBalancer>
    </HTTPTargetConnection>
    <Path>/test</Path>
  </TargetEndpoint>
```

3.14.7.1.2 Maximum Failures

The maximum number of failed requests from the API proxy to the TargetServer that results in the request being redirected to another TargetServer. As configured below, target1 will be removed from rotation after 5 failed requests.

```
  <TargetEndpoint name="default">
    <HTTPTargetConnection>
        <LoadBalancer>
          <Algorithm>RoundRobin</Algorithm>
          <Server name="target1" />
             <MaxFailures>5</MaxFailures>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        <Server name="target2" />
      </LoadBalancer>
      <Path>/test</Path>
  </HTTPTargetConnection>
</TargetEndpoint>
```

> **Note**
>
> If you configure MaxFailures without also configuring a HealthMonitor, then eventually all of your TargetServers will be removed from rotation. If you configure MaxFailures, always configure an associated HealthMonitor.
>
> The MaxFailures default is 0. This means that SAP API Management Edge always tries to connect to the target for each request and never removes the target server from the rotation.

Retry

Retry the request once to a different server after an I/O error (not an HTTP status code 500).

```
<RetryEnabled>true</RetryEnabled>
```

IsFallback

One (and only one) TargetServer can be set as the 'fallback' server. The fallback TargetServer is not included in load balancing routines until all other TargetServers are identified as unavailable by the load balancer. When the load balancer determines that all TargetServers are unavailable, all traffic is routed to the fallback server. For example:

```
<TargetEndpoint name="default">
  <HTTPTargetConnection>
      <LoadBalancer>
        <Algorithm>RoundRobin</Algorithm>
        <Server name="target1" />
        <Server name="target2" />
        <Server name="target3" />
          <IsFallback>true</IsFallback>
      </LoadBalancer>
      <Path>/test</Path>
  </HTTPTargetConnection>
</TargetEndpoint>
```

The configuration above results in round robin load balancing between targets 1 and 2 until both targets 1 and 2 are unavailable. When targets 1 and 2 are unavailable, all traffic is routed to target 3.

Path

Path defines a URI fragment that will be appended to all requests issued by the TargetServer to the backend server.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.14.8 Configuring a target server for SSL

If you are using a TargetServer to define the backend service, and the backend service requires the connection to use the HTTPS protocol, then you must enable SSL in the TargetServer definition. This is necessary because the <Host> tag does not let you specify the connection protocol. Shown below is the TargetServer definition for one-way SSL where SAP API Management Edge makes HTTPS requests to the backend service:

```
<TargetServer name="target1">
  <Host>weather.yahooapis.com</Host>
  <Port>443</Port>
  <IsEnabled>true</IsEnabled>
  <SSLInfo>
      <Enabled>true</Enabled>
  </SSLInfo>
</TargetServer>
```

If the backend service requires two-way, or mutual, SSL, then you configure the TargetServer by using the same SSL configuration settings as TargetEndpoints:

```
<TargetServer  name="TargetServer 1">
    <IsEnabled>true</IsEnabled>
    <Host>www.example.com</Host>
    <Port>443</Port>
    <SSLInfo>
        <Ciphers/>
        <ClientAuthEnabled>true</ClientAuthEnabled>
        <Enabled>true</Enabled>
        <IgnoreValidationErrors>false</IgnoreValidationErrors>
        <KeyAlias>keystore-alias</KeyAlias>
        <KeyStore>keystore-name</KeyStore>
        <Protocols/>
        <TrustStore>truststore-name</TrustStore>
    </SSLInfo>
</TargetServer >
```

For information on the <SSLInfo> properties, such as <Ciphers>, <ClientAuthEnabled>, etc., see the information on setting those properties for a Virtual Host.

For complete instructions on configuring outbound SSL, see Configuring SSL to the Backend Service.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3.14.8.1 Health Monitoring

Health monitoring enables you to enhance load balancing configurations by actively polling the backend service URLs defined in the TargetServer configurations. The HealthMonitor acts as a simple client that invokes a backend service over TCP or HTTP. A TCP client simply ensures that a socket can be opened. You configure the HTTP client to submit a valid HTTP request to the backend service. You can define HTTP GET, PUT, POST, or DELETE operations. Typically, you define a simple GET request that simply shows that the backend service is available.

You create a HealthMonitor by setting one up in a TargetEndpoint's HTTPConnection configuration. A simple HealthMonitor defines an IntervalInSec combined with either a TCPMonitor or an HTTPMonitor.

TCPMonitor

The configuration below defines a HealthMonitor that polls each TargetServer by opening a connection on port 80 every 5 seconds.

- If the connection fails or takes more than 10 seconds to connect, then the failure count increments by 1 for that TargetServer.
- If the connection succeeds, then the failure count for the TargetServer is reset to 0.

You can add a HealthMonitor as a child element of the TargetEndpoint's HTTPTargetConnetion element, as shown below:

```
<TargetEndpoint name="default">
  <HTTPTargetConnection><HealthMonitor>
    <IsEnabled>true</IsEnabled>
    <IntervalInSec>5</IntervalInSec>
    <TCPMonitor>
      <ConnectTimeoutInSec>10</ConnectTimeoutInSec>
      <Port>80</Port>
    </TCPMonitor>
  </HealthMonitor>
```

HealthMonitor with TCPMonitor Configuration Elements

Name	Description	Default	Required?
IsEnabled	A boolean that enables or disables the HealthMonitor.	false	No
IntervalInSec	The time interval, in seconds, between each polling TCP request.	0	Yes
ConnectTimeoutInSec	Time in which connection to the TCP port must be established to be considered a success. Failure to connect in the specified interval counts	0	Yes

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Default	Required?
	as a failure, incrementing the load balancer's failure count for the TargetServer.		

HTTPMonitor

A sample HealthMonitor that uses an HTTPMonitor will submit a GET request to the backend service once every five seconds. The sample below adds an HTTP Basic Authorization header to the request message. The Response configuration defines settings that will be compared against actual response from the backend service. In the example below, the expected response is an HTTP response code 200 and a custom HTTP header ImOK whose value isYoureOK. If the response does not match, then the request will treated as a failure by the load balancer configuration.

> **Note**
>
> All of the Request and Response settings in an HTTP monitor will be specific to the backend service that must be invoked.

```
<HealthMonitor>
  <IsEnabled>true</IsEnabled>
  <IntervalInSec>5</IntervalInSec>
  <HTTPMonitor>
    <Request>
      <ConnectTimeoutInSec>10</ConnectTimeoutInSec>
      <SocketReadTimeoutInSec>30</SocketReadTimeoutInSec>
      <Port>80</Port>
      <Verb>GET</Verb>
      <Path>/healthcheck</Path>
      <Headers>
        <Header name="Authorization">Basic 12e98yfw87etf</Header>
      </Headers>
    </Request>
    <SuccessResponse>
      <ResponseCode>200</ResponseCode>
      <Headers>
        <Header name=”ImOK”>YoureOK</Header>
      </Headers>
    </SuccessResponse>
  </HTTPMonitor>
</HealthMonitor>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

HealthMonitor with HTTPMonitor Configuration Elements

Name	Description	Default	Required?
IsEnabled	A boolean that enables or disables the HealthMonitor.	false	No
IntervalInSec	The time interval, in seconds, between each polling HTTP request.	0	Yes
Request	Configuration options for the outbound HTTP request message sent by the HealthMonitor to the TargetServers in the rotation.	N/A	Yes
ConnectTimeoutInSec	Time, in seconds, in which the TCP connection handshake to the HTTP service must complete to be considered a success. Failure to connect in the specified interval counts as a failure, incrementing the LoadBalancer's failure count for the TargetServer.	0	No
SocketReadTimeoutInSec	Time, in seconds, in which data must be read from the HTTP service to be considered a success. Failure to read in the specified interval counts as a failure, incrementing the LoadBalancer's failure count for the TargetServer.	0	No
Port	The port on which the HTTP connection to the backend service will be established.	N/A	No
Verb	The HTTP verb used for each polling HTTP request to the backend service.	N/A	No

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Default	Required?
Path	The path appended to the URL defined in the TargetServer. Use the path element to configure a 'polling endpoint' on your HTTP service.	N/A	No
Headers	A list of one or more HTTP headers and their values that will be populated on each polling HTTP request.	N/A	No
Payload	The HTTP body generated for each polling HTTP request. Note that this element is not required for GET requests.	N/A	No
Response	Matching options for the inbound HTTP response message generated by the polled backend service. Responses that do not match increment the failure count by 1.	N/A	Yes
ResponseCode	The HTTP response code expected to be received from the polled TargetServer. A code different than the code specified results in a failure, and the count being incremented for the polled backend service. You can define multiple ResponseCode elements.	N/A	No
Headers	A list of one or more HTTP headers and values expected to be received from the polled backend service. Any HTTP headers or values on the response that are different from those	N/A	No

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Default	Required?
	specified result in a failure, and the count for the polled TargetServer is incremented by 1. You can define multiple Header elements.		

3.14.8.2 Configuring outbound client SSL for mutual authentication

TargetServers expose the same SSL configuration settings as TargetEndpoints.

```
<TargetServer  name="TargetServer 1">
    <IsEnabled>true</IsEnabled>
    <Host>www.example.com/Host>
    <Port>443</Port>
    <SSLInfo>
        <Ciphers/>
        <ClientAuthEnabled>true</ClientAuthEnabled>
        <Enabled>true</Enabled>
        <IgnoreValidationErrors>false</IgnoreValidationErrors>
        <KeyAlias>keystore-alias</KeyAlias>
        <KeyStore>keystore-name</KeyStore>
        <Protocols/>
        <TrustStore>truststore-name</TrustStore>
    </SSLInfo>
</TargetServer >
```

For complete instructions on configuring outbound client SSL, see Configuring SSL to the Backend Service.

3.14.9 Virtual Hosts

Virtual hosts let multiple domain names connect to the same host. A virtual host on SAP API Management defines the domains and Router ports on which an API proxy is exposed, and, by extension, the URL that apps use to access an API proxy. A virtual host also defines whether the API proxy is accessed by using the HTTP protocol, or by the encrypted HTTPS protocol. In SAP API Management Edge, a virtual host is identified by a name. By default, when you first create an SAP API Management Edge organization, you get two virtual hosts in each environment:

- `default - http://[org]-[environment].sap.net (port 80)`
- `secure - https://[org]-[environment].sap.net (port 443)`

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

One of the main uses of virtual hosts is to let you customize the domain name of your APIs hosted on SAP API Management. For example, on a cloud-based version of SAP API Management, the default domain name of an organization called "myorg" in the prod environment is "myorg-<host>:<port>/". Therefore, to access an API in that organization, a developer uses a URL in the form:

```
https://myorg-prod.<host>:<port>/v1/{proxy-base-path}/{resource-path}
```

However, a domain name containing "host:port" may not be what you want to expose to your customers. A virtual host can map your own domain name to your organization, letting developers access your API through a domain specific to your company, for example:

```
https://api.myCompany.com/v1/{proxy-base-path}/{resource-path}
```

Note that creating a virtual host does not also make the domain name accessible. You must still create a DNS record for the domain.

There are no default organizations, environments, or virtual hosts created for you. After you complete the SAP API Management installation process, your first action is typically to create an organization, environment, and virtual host through the "onboarding" process.

Note:

To perform onboarding, please check section 3.7 Onboard an Organization topic in the SAP API Management Installation Onboarding Guide.

where <inst-root> is the root directory of your installation.

This script prompts you to create a user, organization, environment, and virtual host.

For example, if you use the default values in the setup-org.sh script, you create:

- A user of your choosing to function as the organization administrator
- An organization named example
- An environment in the organization named prod
- A virtual host in the environment named default that allows HTTP access on port 9001
- No host alias

After running the onboarding script, you can later add any number of organizations, environments, and virtual hosts to your installation.

Virtual hosts are opened on the Router node. Therefore, you have to ensure that the port that you specify for the virtual host is open on the Router node. You can use a command in the form below to open a port:

```
$ iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 9001 -j ACCEPT --
verbose
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

After running that script, you can access your APIs by using a URL in the form:

```
http://<router-ip>:9001/{proxy-base-path}/{resource-path}
```

Note that this URL uses the IP address of the Router and the virtual host port on the Router to access your APIs. Typically, you do not publish your APIs to customers with an IP address and port number. Instead, you define a DNS entry for the router and port. For example:

```
http://myAPI.myCo.com/{proxy-base-path}/{resource-path}
```

If you define the DNS entry, then you also must create a host alias for the virtual host that matches the domain name of the DNS entry. From the example above, you would specify a host alias of myAPI.myCo.com when you create the virtual host.

Note also that the URL uses the HTTP protocol. To use the encrypted HTTPS protocol, you must create a virtual host that references a keystore. A keystore contains the certificate and private key required to support HTTPS.

3.14.9.1 Viewing information about a virtual host

You can use the SAP API Management UI to see the virtual hosts defined in an environment:

1. Login to the management UI at http:// <ms-ip>:9000 , where <ms-ip> is the IP address of the Management Server node.
2. In the management UI menu, select APIs > Environment Configuration.
3. Select the environment, such as prod or test. The virtual hosts defined for the environment appear.

If the virtual host is configured to use a keystore or truststore, select the Show button to see more information.

You can also use the SAP API Management APIs to view information about virtual hosts. For example, the List Virtual Hosts API returns a list of all virtual hosts:

```
$ curl -X GET -H "accept:application/xml" \
https://<host:port>/v1/o/{org}/environments/{env}/virtualhosts \
-u myname:mypass
```

Sample response:

```
[
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
 "default",
 "secure"
]
```

To see information about a specific virtual host, use the Get Virtual Host API:

```
$ curl -X GET -H "accept:application/xml" \
https://<host:port>/v1/o/{org_name}/environments/{env_name}/virtualhosts/default \
-u myname:mypass
```

Sample response:

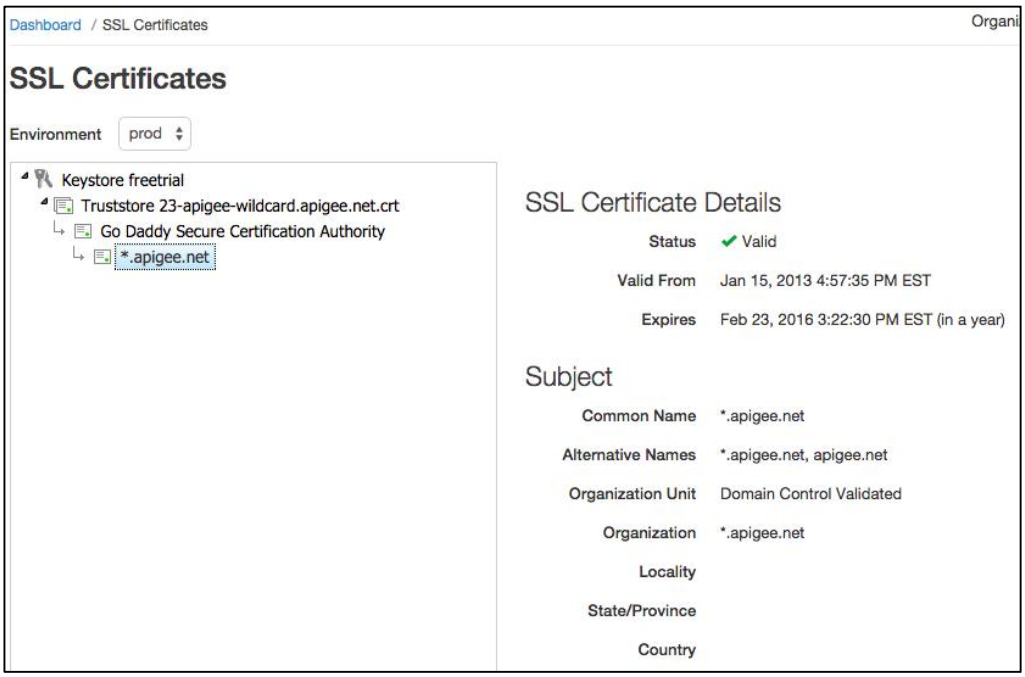
```
{
 "hostAliases" : [sapdocs-prod.<host:port> ],
 "interfaces" : [ ],
 "name" : "default",
 "port" : "9001"
}
```

If the virtual host is configured to use SSL, a lock icon appears next to the name of the virtual host. That means an SSL certificate, key, and certificate chain has been uploaded to SAP API Management and associated with the virtual host.

To see information about the available certificates:

1. Login to the management UI.
2. In the SAP API Management UI menu, select Admin > SSL Certificates.
3. Expand the keystores until you see the certificate.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.



To use the the Get a Cert from a Keystore or Truststore to view a cert:

```
$ curl -H 'accept: application/xml' \
https://<host:port>/v1/o/{org_name}/environments/{env_name}/keystores/freetrial/certs/
23-sap-wildcard.<host:port>.crt \
-u myname:mypass
```

3.14.9.2 Updating an API proxy after creating a virtual host

An API proxy references the virtual hosts that it supports in its ProxyEndpoint definition. When you create a new API proxy, SAP API Management automatically configures its ProxyEndpoint to use all available virtual hosts.

After SAP API Management notifies you that your new virtual host has been created, any new API proxies that you create automatically reference the virtual host.

> **Note**
>
> If you create a new API proxy that should not be accessible over a particular virtual host, then you must edit the API proxy to remove that virtual host from its ProxyEndpoint.

If you created any API proxies before requesting the virtual host, then you must edit the API proxy to add the new virtual hosts to its ProxyEndpoint. Otherwise, the API proxy is not accessible by the virtual host.

If you are developing your API proxies in XML, edit the XML file for the ProxyEndpoint to add or remove a virtual host.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

To use the SAP API Management UI to edit the ProxyEndpoint:

1. Login to the SAP API Management UI.
2. In the SAP API Management UI menu, select APIs.
3. Select the name of the API proxy to update.
4. Select the Develop tab.
5. Under Proxy Endpoints, select default.
6. In the code area:
    1. Remove any <VirtualHost> elements for virtual hosts not supported by the API proxy.
    2. Add a new <VirtualHost> element with the name of the new virtual host. For example, if the new virtual host is named MyVirtualHost, add the following tag:

```
<HTTPProxyConnection>
   <BasePath>/oauth10a/client_credential</BasePath>
   <VirtualHost>default</VirtualHost>
   <VirtualHost>secure</VirtualHost>
   <VirtualHost>MyVirtualHost</VirtualHost>
</HTTPProxyConnection>
```

7. Save the API proxy. If the API proxy has been deployed, saving it redeploys it with the new setting.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4 Secure APIs

4.1 Org Administration

4.1.1 What are organization users?

Organization users are given explicit permission by the organization administrator to create, read, edit, and/or delete entities in an SAP API Management organization. Permissions are role-based, where a role conveys a specific, targeted set of permissions. This permission scheme is also called role-based access control, or RBAC for short.

Organization users are typically members of your API team who develop and test APIs, run reports, and perform other API admin tasks. Do not confuse organization users with app developers, the consumers of your APIs. The process of onboarding app developers and managing their access to your APIs is an entirely separate topic.

Also, note that topic applies to API management, not API BaaS, which has its own user management framework.

4.1.2 What SAP API Management entities do organization users work with?

Organization users can interact with the following entities. The degree of interaction permitted depends on the role or roles that are assigned to the user by the organization administrator.

- API proxies
- API products
- Developer apps
- Developers
- Environments (Trace tool sessions and deployments)
- Custom reports (Analytics)

4.1.3 Manage Users and Roles

Before you and your team can start using SAP API Management, you need to have an account and an organization. Once you have an account and an organization, if you're an administrator, you have access to the Admin tab where you can add and modify users. As part of adding or modifying users, you set the user's role.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.1.3.1 Viewing User Data

The Organization Users table on the Admin > Organization Users page lists all of the users attached to the current organization. For each user you can see:

> **Note**
>
> By default, all users associated with an organization can view details about other organization users, such as email address, first name, and last name. Only users with the Organization Administrator role can add or update other organization users.

- Name: The name of the user you entered when you created the user.
- Primary email: The email address you entered when you created the user.
- Role: The role of the user, which determines the degree of access. By default, all users have a user role that gives them full access to all features in SAP API Management.

4.1.3.2 Adding Users

Users of API Platform are members of the API team who develop and test the API, or run reports—not external developers. To add an organization user:

1. In the SAP API Management management UI, while logged in as an organization administrator, select Admin > Organization Users.
2. Choose + User. The "Add a User" screen appears.
3. Enter the user's First Name, Last Name, and Email.
4. Select the Role you want to offer to the users.
5. Choose Save.

The First Name, Last Name, and Email Address fields are editable, so if needed, you can change what you initially entered for the user. You can also change the role selection if needed.

4.1.3.3 Adding Roles to a User

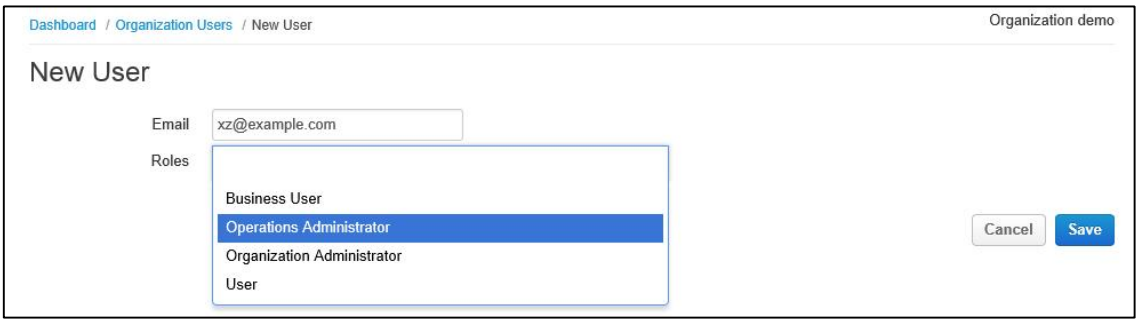
You can add one or more roles to a user when you create a new user or if you edit an existing user.

> **Note**
>
> If a user has multiple roles assigned, the greater permission takes precedence. For example, if one role doesn't allow the user to create API proxies, but another role does, then the user can create API proxies. In general, it is not a common use case to assign users multiple roles.

1. Select Admin > Organization Users.
2. Either choose + User or choose an existing user.
3. Choose in the Roles field, and a dropdown appears.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4. Select a role to add.
5. Repeat steps 3 and 4 to add additional roles to the user if you want.
6. Choose Save.



4.1.3.4 Removing Users from an Organization

To remove a user from an organization, you must be an org administrator.

1. Select the user in the Organization Users table.
2. Choose Remove.

> **Note**
>
> This only removes the user from the current account. If the user is a member of multiple accounts, they remain in the system.

4.1.3.5 Editing a user's profile and role

An org admin can edit the first name, last name, and email address fields in the management UI.

Administrators can also change the user's role.

4.1.4 Assign Roles

This topic discusses role-based access control for SAP API Management organizations and explains how to create roles and assign users to them. You must be an organization administrator to perform the tasks described here.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.1.4.1 What are roles?

Roles are essentially CRUD-based permission sets. CRUD means "create, read, update, delete". For example, a user may be given a role that permits her to read or "get" details about a protected entity, but not permission to update or delete it. The organization administrator is the highest-level role, and can perform any CRUD operation on protected entities, which include:

- API proxies
- API products
- Developer apps
- Developers
- Environments (Trace tool sessions and deployments)
- Custom reports (Analytics)

4.1.4.2 Getting started

You must be an org administrator

You must be an SAP API Management organization administrator to create users and assign roles. Only organization admins can see and use the Admin menu, which is for managing users and roles.

Users must have SAP API Management accounts

Before you can add an organization user to your org and assign roles, that user must have an SAP API Management account.

4.1.4.3 What you need to know about user roles

In SAP API Management, user roles form the basis of role-based access, meaning that you can control what functions a person can access by assigning them a role (or roles). Here are a few things you need to know about roles:

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- When you create your own account, your role is set automatically to organization administrator in your organization. If you add users to your organization, you set the user role (or roles) at the time that you add them.
- When an org admin adds you to an org, your role (or roles) are determined by the administrator. The organization administrator can later change your role(s) if necessary.
- Users can be assigned more than one role. If a user has multiple roles assigned, the greater permission takes precedence. For example, if one role doesn't allow the user to create API proxies, but another role does, then the user can create API proxies. In general, it is not a common use case to assign users multiple roles.
- By default, all users associated with an organization can view details about other organization users, such as email address, first name, and last name.

It's important to understand that user roles are specific to the organization in which they were assigned. An SAP API Management user can belong to multiple organizations, but roles are organization-specific. For example, a user can be have the organization administrator role in one org and only the user role in another.

4.1.4.4 SAP API Management Edge built-in roles

Each SAP API Management Edge organization comes with a few built-in roles that you can assign to administrative users:

- Organization Administrator - Super user. Has full CRUD access to resources in the organization. In an SAP API Management Edge installation, the most powerful role is the System Administrator Role, which also has access to system-level functions that the Organization Administrator doesn't.
- Read-only Organization Administrator - Has read-only access to resources in the organization.
- Operations Administrator - Deploys and tests APIs; has read-only access to other resources.
- Business User - Creates and manages API products, developers, developer apps, and companies; creates custom reports on API usage; has read-only access to other resources.
- User - Creates API proxies and tests them in the test environment; has read-only access to other resources.

The built-in roles control the level of access in both the management UI and the management API.

To see the permissions set for each built-in role (as an Organization Administrator or a Read-only Organization Administrator), select Admin > Organization Roles > name_of_role in the management UI.

For more information about how resource paths map to permissions, see "About permission setting" in Creating roles with the API.

4.1.4.4.1 About the Developer Administrator role

When a Developer Portal is provisioned for you, another role is added to your organization called Developer Administrator. This role, which includes a single user called devadmin+{org_name}@apigee.com, is solely for the purpose of connecting your Developer Portal to your SAP API Management Edge organization. Because the portal displays your SAP API Management Edge developer apps, API products, and so on, it must stay in sync with your SAP API Management Edge organization by making management API calls that require authentication. The devadmin "user" has the necessary permissions.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The connection between the portal and your SAP API Management Edge organization is configured in Drupal under Configuration > Dev Portal in the Drupal admin menu (you must be logged into Drupal as an administrator). If the Connection Status is shown as "Connection Successful," you won't see the devadmin user in the configuration. However, if the connection is shown as failed, you would enter the devadmin email address and password to reestablish the connection.

Don't add real users to the Developer Administrator role. (In fact, the management UI and API shouldn't let you.) That role should contain only the devadmin user.

4.1.4.5 Adding roles to users

You can add one or more roles to a user when you create a new user or if you edit an existing user.

> **Note**
>
> If a user has multiple roles assigned, the greater permission takes precedence. For example, if one role doesn't allow the user to create API proxies, but another role does, then the user can create API proxies. In general, it is not a common use case to assign users multiple roles.

1. Select Admin > Organization Users.
2. Either choose + User or choose an existing user.
3. Choose in the Roles field, and a dropdown appears.
4. Select a role to add.
5. Repeat steps 3 and 4 to add additional roles to the user if you want.

4.1.4.6 Default role permissions

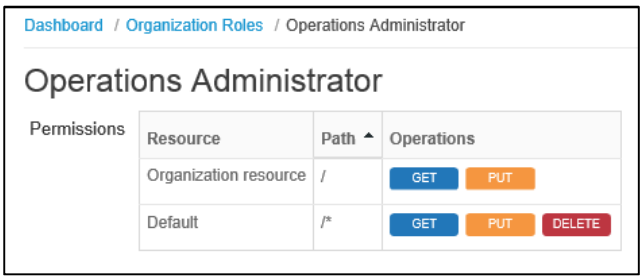
SAP API Management provides a set of default roles, out-of-the-box, and they are listed below.

- Business user
- Operations administrator
- Organization administrator
- User

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

If you are an organization admin

Organization admins can see the entire list of permissions for each type of user. Just go to Admin > Organization Roles. When you choose on a role, it takes you to a table that looks like this:



The table shows you the levels of protection for resources. In this context, resources refer to "entities" that users can interact with through the SAP API Management UI and API.

- The first column lists the general names of resources that users interact with. It also includes some other things like API Proxies, Products, Deployments, etc. This column reflects the names of things as you see them in the management UI.
- The second column lists the paths used to access resources through the management API.
- The third column lists the operations the role can perform on each resource and path. The operations are GET, PUT, and DELETE. In the UI, these same operations are referred to as View, Edit, and Delete. Just keep in mind that the UI and API uses different terms for these operations.

If you are not an org admin

Although you are not permitted to add or change a user's roles or view the role properties in the UI, you can choose a role below to see a page that lists the permissions granted to that role.

- Business user
- Operations administrator
- Organization administrator
- Read-only organization administrator
- User

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.1.4.7 Role operations

You can assign roles through management APIs or through the management UI. Either way, you're working with CRUD permissions, although the API and UI use slightly different terminology.

SAP API Management APIs allow these CRUD operations:

- GET: Enables a user to view a list of protected resources or to view a singleton RBAC resource
- PUT: Enables a user to create or update a protected resource (encompassing both PUT and POST HTTP methods)
- DELETE: Enables a user to delete an instance of a protected resource. Note: You can only delete a specific instance of a resource. You cannot, for example, delete ALL API proxies, only a specific one.

The management UI refers to the same CRUD operations, but with different wording:

- View: Enables a user to view protected resources. Typically, you can view resources one at a time, or view a list of resources.
- Edit: Enables a user to update a protected resource.
- Create: Enables a user to create a protected resource.
- Delete: Enables a user to delete an instance of a protected resource. Note: You can only delete a specific instance of a resource. You cannot, for example, delete ALL API proxies, only a specific one.

4.1.4.8 Creating custom roles

Custom roles let you apply fine-grained permissions to these SAP API Management entities such as API proxies, products, developer apps, developers, and custom reports.

4.1.5 Creating custom roles in the UI

This topic explains how to create custom roles in the management UI. Only an organization administrator can create custom roles. Custom roles allow the admin to fine-tune which permissions to grant on specific resources. For example, you can create a custom role that only grants users permission to view certain resources.

> **Note**
>
> SAP API Management also includes a set of pre-defined roles. For an introduction to roles and the permissions granted to roles, see Manage users and roles. We recommend that you review that topic before creating custom roles.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.1.5.1 What are custom roles?

You can create custom roles to fine-tune access to these SAP API Management entities:

- API proxies
- API products
- Developer apps
- Developers
- Environments (Trace tool sessions and deployments)
- Custom reports (Analytics)

You can achieve even more granularity by applying role based access to specific instances of an entity. For example, you can apply role-based access to all API products or to specific ones.

4.1.5.2 Precedence

More granular permissions take precedence over less granular ones. For example, permissions applied to a specific developer app take precedence over a less-granular permission applied to all developer apps.

4.1.5.3 Entity collections and instances

You can set some custom role permissions on an entity as a collection (e.g., all API products) or on a single instance of an entity (e.g., one specificproduct).

If you set permissions on an instance, a privileged user can perform the permitted operations on that instance only. If set on a collection (e.g., all API proxies), the user can perform the operations on any instance in the collection.

You can also enable deploy and trace options on both collections and instances of APIs and caches. These operations are also environment specific. That is, you can allow a role to deploy only to the prod environment.

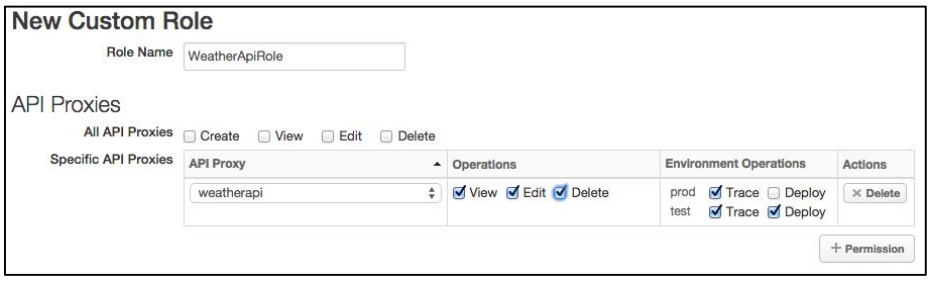
4.1.5.4 Using the management UI to create custom roles

An org administrator can create custom roles through the management UI.

1. Go to Admin > Organization Roles.
2. Choose + Custom Role.
3. Use the New Custom Role dialog to create the custom role.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The following screen shot shows part of the New Custom Role dialog. For example, this role is called WeatherApiRole, and it allows a user to view, edit, and delete an API proxy with the path /weatherapi. In addition, this user can view trace sessions in both prod and test environements, but can only deploy to the test environment.



4.1.6 Creating roles with the API

This topic discusses how to create custom roles and assign roles to users through the management API. We also show how to test role assignments through the API.

4.1.6.1 Create a new custom role

Create a 'development' role to enable developers to view, create, and update API proxies.

```
$ curl -u myname:mypass https://<host:port>/v1/o/{org_name}/userroles -H "Content-type:application/json" -X POST -d'{ "role" : [ { "name" : "development" } ] }'
```

4.1.6.2 Add permissions to development role

The permissions that can be set on collections are GET and PUT.

The permissions that can be set on individual members of a collection are GET, PUT, and DELETE. For the 'development' role, we add GET and PUT permissions on APIs. GET enables users to view any APIs, including API proxy configuration files, associated policies, Javascript, XSLT files, and so on. The PUT permission on APIs enables developers to create, modify, import, export, deploy and undeploy API proxies.

The path attribute specifies the resource on which you set the permissions. For example, /applications, /apps, /apiproducts, /developers, or /reports.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
curl -u myname:mypass
https://<host:port>/v1/o/{org_name}/userroles/development/permissions -H "Content-
type:application/json" -X POST -d'{"path" : "/applications","permissions" : [ "post",
"get" ]}'
```

4.1.6.3 Create a role: testing

Create a 'testing' role to enable quality engineers to view API proxies and their contents (including, for example, policies).

```
$ curl -u myname:mypass https://<host:port>/v1/o/{org_name}/userroles -H "Content-
type:application/json" -X POST -d'{ "role" : [ { "name" : "testing" } ] }'
```

4.1.6.4 Add permissions to testing role

GET enables users to view any APIs, including their configuration files, as well as any associated policies, JavaScript, XSLT files, and so on. By adding this permission to the 'testing' role, we enable quality engineers to view the contents of the APIs that they are testing. Users in this role will not, however, be able to create, modify, import, export, deploy and undeploy API proxies.

```
$ curl -u myname:mypass
https://<host:port>/v1/o/{org_name}/userroles/testing/permissions -H "Content-
type:application/json" -X POST -d'{"path" : "/applications","permissions" : [ "get"
]}'
```

The minimum set of permissions that have to be set to allow the user to log in to the SAP API Management UI are:

```
{"path" : "/,"permissions" : [ "get" ]}'
{"path" : "/*","permissions" : [ ]}'
{"path" : "/environments","permissions" : [ "get" ]}
{"path" : "/userroles","permissions" : [ "get" ]}
```

Use the following cURL command to set these permissions:

```
curl -H "Content-Type:application/json" -u sgilson@sap.com:refLecT9 \
-X POST \
http://<ms-IP>:8080/v1/organizations/{org_name}/userroles/testing/resourcepermissions
\
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
-d '{
 "resourcePermission" : [
   {
    "path" : "/",
    "permissions" : [ "get" ]
   },
{
    "path" : "/*",
    "permissions" : []
   },
{
    "path" : "/environments",
    "permissions" : [ "get" ]
   },
   {
    "path" : "/userroles",
    "permissions" : [ "get"]
   }
  ]
}'
```

where <ms-IP> is the IP address or DNS name of the SAP API Management Server.

4.1.6.5 Add user to testing role

To provision a user with a user role:

```
$ curl -u myname:mypass
https://<host:port>/v1/o/{org_name}/users/justauser@sap.com/userroles -H "Content-
type:application/json" -X POST -d'{"role" : [ {"name" : "testing"} ] }'
```

4.1.6.6 View APIs as user

Impersonate the user and make a request to API Services to view API proxies. The user should be able to view APIs, along with their contents.

```
$ curl -u justauser@sap.com:secret https://<host:port>/v1/o/{org_name}/apis
$ curl -u justauser@sap.com:secret
https://<host:port>/v1/o/{org_name}/apis/{api_name}/policies
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.1.6.7 Create API as user in testing role

Impersonate the user and make a request to API Services to create an API proxy. The request will be rejected by API Services, as the role 'testing' does not permit the user to create APIs.

```
$ curl -u justauser@sap.com:secret -H "Content-Type: application/json"
https://<host:port>/v1/o/{org_name}/apis -X POST -d'{"name" : "rbacTestApi"}'
```

4.1.6.8 Add user to development role

Now provision the user with the 'development' role.

```
$ curl -u myname:mypass
https://<host:port>/v1/o/{org_name}/users/justauser@sap.com/userroles -H "Content-
type:application/json" -X POST -d'{"role" : [ {"name" : "development"} ] }'
```

4.1.6.9 Create API as user in development role

Impersonate the user and repeat the request to the API Platform to create an API proxy. The request will be successful, as the role 'development' does permit the user to create APIs.

```
$ curl -u justauser@sap.com:secret -H "Content-Type: application/json"
https://<host:port>/v1/o/{org_name}/apis -X POST -d'{"name" : "rbacTestApi"}'
```

4.1.6.10 Get user roles for a user

As an organization administrator, you can check the list of user roles for a user at any time:

```
$ curl -u myname:mypass
https://<host:port>/v1/o/{org_name}/users/justauser@sap.com/userroles
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2 OAuth

This topic offers a basic overview of OAuth 2.0 on SAP API Management.

4.2.1 OAuth 2.0

4.2.1.1 Introduction to OAuth 2.0

There are many books, blogs, and sites devoted to OAuth 2.0. We highly recommend that you begin by reviewing the IETF OAuth 2.0 specification. Here's the definition of OAuth 2.0 from the OAuth 2.0 IETF specification itself:

"The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf."

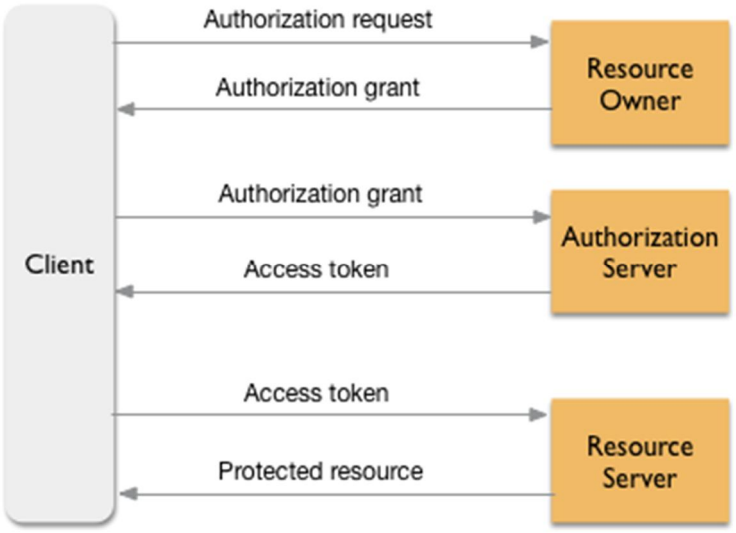
The main thing you need to know is that OAuth 2.0 provides a way for apps to gain limited access to a user's protected resources (think of bank account or any other sensitive information a user might wish to access from an app) without the need for the user to divulge her login credentials to the app.

> **Note**
>
> Let's say you found an app that lets you access multiple bank accounts. Maybe the app is used to help with your taxes, or for managing your budget. You have three bank accounts that you want to manage with the app, and you have a different username and password for each one. Now, for the app to access your bank accounts, it needs those sensitive credentials. Basically, you're giving this sensitive information to the app. Do you trust the app to possess your passwords and usernames? What if the app is hacked? Now, you've got to remember change your bank credentials for each account. This is the fundamental problem that OAuth 2.0 solves. Read on to learn how.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.1.1 The OAuth 2.0 flow

Here is the general flow for the OAuth 2.0 security framework. We'll discuss this flow in more detail in this topic, starting with a diagram, which illustrates a lot about how OAuth 2.0 works. If you're unfamiliar with the terms used in this diagram, read this section for a quick introduction.



Terms you should know

- Client -- Also called "the app". It can be an app running on a mobile device or a traditional web app. The app makes requests to the resource server for protected assets on behalf of the resource owner. The resource owner must give the app permission to access the protected resources.
- Resource owner -- Also called an "end user". This is generally the person (or other entity) who is capable of granting access to a protected resource. For example, if an app needs to use data from one of your social media sites, then you are the resource owner -- the only person who can grant the app access to your data.
- Resource server -- Think of the resource server as a service like Twitter or Facebook, or an HR service on your intranet, or a partner service on your B2B extranet. SAP API Management is a resource server whenever OAuth token validation is required to process API requests. The resource server needs some kind of authorization before it will serve up protected resources to the app.
- Authorization server -- The authorization server is implemented in compliance with the OAuth 2.0 specification, and it is responsible for validating authorization grants and issuing the access tokens that give the app access to the user's data on the resource server. You can configure "token endpoints" on SAP API Management, in which case it takes on the role of authorization server.
- Authorization grant -- Gives the app permission to retrieve an access token on behalf of the end user. OAuth 2.0 defines four specific "grant types".
- Access token -- A long string of characters that serves as a credential used to access protected resources.
- Protected resource -- Data owned by the resource owner. For example, the user's contact list, account information, or other sensitive data.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.1.2 Where SAP API Management fits in

You can protect any API proxies through SAP API Management with OAuth 2.0. It includes an authorization server implementation, and as such, can generate and validate access tokens. Developers begin by registering their apps with SAP API Management. Registered apps can request access tokens through any of the four grant type interactions.

SAP API Management provides a multi-faceted OAuthV2 policy that implements the details of each grant type, making it relatively easy to set up OAuth on SAP API Management. For example, you can configure a policy that receives a request for an access token, evaluates all required credentials, and returns an access token if the credentials are valid.

If you want to see how it fits together, check out the end-to-end tutorial. The tutorial steps through the process of configuring the OAuth 2.0 policy, requesting an access token, and calling a protected API.

Note that any resource servers that your secure API proxy calls should be behind a firewall (that is, the resources must not be accessible through any means besides the API proxy or another API that is well secured).

4.2.1.1.3 What are OAuth 2.0 grant types?

Think of grant types as different paths or interactions an app can take to gain an access token. Each grant type addresses one or more use cases, and you'll need to select which grant type(s) to use based on your own needs. In general, each grant type has advantages and disadvantages, and you'll need to weigh the tradeoffs based on your business use cases. One important consideration is the "trustworthiness" of the apps that will be accessing your data. Generally, third-party apps are less trustworthy than apps that are developed and used within an enterprise.

SAP API Management supports the four main OAuth 2.0 grant types:

- authorization code -- Considered the most secure grant type. Before the authorization server issues an access token, the app must first receive an authorization code from the resource server. You've seen this flow anytime your app opens a browser to the resource server's login page and invites you log in to your actual account (for example, Facebook or Twitter).

  If you successfully log in, the app will receive an authorization code that it can use to negotiate an access token with the authorization server. Typically, this grant type is used when the app resides on a server rather than on the client. This grant type is considered highly secure because the client app never handles or sees the user's username or password for the resource server (that is, for example, the app never sees or handles your Twitter credentials). This grant type flow is also called "three-legged" OAuth.
- implicit -- Considered a simplified version of authorization code. Typically this grant type is used when the app resides on the client. For example, the app's code is implemented in a browser using JavaScript or another scripting language (instead of residing and running on a separate web server). In this grant type flow, the authorization server returns an access token directly when the user is authenticated, rather than issuing an authorization code first. Implicit grants can improve app responsiveness in some cases, but this advantage needs to be weighed against possible security implications as described in the IETF specification.
- resource owner password credentials -- In this flow, the client is issued an access token when the user's username/password are validated by the authorization server. This flow is recommended for highly trusted applications. An advantage of this flow over, say, basic authentication, is that the user only presents his username/password once. From then on, the access token is used.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- client credentials -- Consider using for situations where the client app is acting on its own behalf. That is, the client is also the resource owner. This grant type is typically used when the app needs to access a backend data storage service, for example. The app needs to use the service to do its work, and the service is otherwise opaque to the end user. With this grant type, an app can receive an access token by presenting it's client ID and client secret keys to the authorization server. No further steps are required. SAP API Management provides an out-of-the-box client credentials solution that's easy to implement for any API proxy.

4.2.1.1.4 What is an access token?

An access token is a long string of characters that serves as a credential used to access protected resources. Resources tokens (also called bearer tokens) are passed in Authorization headers, like this:

```
$ curl -H "Authorization: Bearer UAj2yiGAcMZGxfN2DhcUbl9v8WsR" \
  http://myorg-test.<host:port>/v0/weather/forecastrss?w=12797282
```

The resource server understands that the access token "stands in" for credentials like username and password. In addition, the access tokens can be issued with restrictions, so that, for example, the app can read but not write or delete data on the resource server. Note that an access token can be revoked if, for instance, the app is compromised. In this case, you will need to get a new access token to continue using the app; however, you will not have to change your username or password on the protected resources server (for example, Facebook or Twitter).

Access tokens generally have an expiration (for security reasons). Some grant types allow the authorization server to issue an refresh token, which allows the app to fetch a new access token when the old one expires. For more details on access and refresh tokens, refer to the IETF OAuth 2.0 specification.

4.2.1.1.5 Limited Access Through Scopes

Through the mechanism of scopes, OAuth 2.0 can grant an app limited access to protected resources. For example, an app may have access only to specific resources, may be able to update resources, or may only be granted read-only access. Under so-called "three-legged" OAuth flows, the user typically specifies the level of access through a consent page (for example, a web page where the user selects the scope with a checkbox of other mechanism).

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.1.6 Registering an app

All clients (apps) must register with the OAuth 2.0 authorization server from which they intend to request access tokens. When you register an app, you receive back a set of keys. One is a public key called the client identifier, and the other is a secret key called the client secret. Without these keys, an app cannot issue requests for authorization codes or access tokens to the authorization server. Note that while the IETF OAuth specification calls these keys client ID and client secret, the SAP API Management UI calls them the Consumer ID and Consumer secret. They are equivalent.

4.2.1.1.7 Summary of OAuth 2.0 use cases

Which OAuth 2.0 grant type flow you chose to implement depends on your specific use case, as some grant types are more secure than others. Your choice of grant types depends on the trustworthiness of the client app and requires very careful consideration, as described in the following table:

Use Case	Trustworthiness	Suggested OAuth 2.0 Authorization Grant Types	Description
B2B (extranet), intranet, other	Highly trusted apps, written by internal developer or developers with a trusted business relationship with the API provider. Apps that need to access resources on their own behalf.	Client credentials	Typically, the app is also the resource owner Requires Client ID and Client secret keys Requires app to be registered with service provider
Intranet sites, portals	Trusted apps written by internal or trusted third-party developers. A good example is logging in to your company HR site to make insurance selections, submit reviews, or change personal information.	Password Implicit	Requires client Id and secret, plus username and password Requires app to be registered with service provider
Publicly available apps	Untrusted apps are written by third-party developers who do not have a trusted business relationship with the API provider. For example,	Authorization code	Requires user to log in to third-party resource provider

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

	developers who register for public API programs should not generally be trusted.		(e.g., Twitter, Facebook) App never sees user's username and password Requires app to be registered with service provider
B2C	There is an individual end user (mobile user) involved, and user credentials are stored on the mobile device.	Implicit	Requires app to be registered with the service provider. User credentials are stored on the device running the app.

4.2.1.1.8 OAuth 2.0 vs. API key security

It is relatively simple to 'step up' your API security scheme from API key validation to OAuth client credentials. Both schemes use the same consumer key and secret to validate the client app. The difference is that client credentials provides an extra layer of control, since you can easily revoke an access token when needed, without requiring you to revoke the app's consumer key. To work with the default OAuth endpoints, you can use any consumer key and secret generated for app in your organization to retrieve access tokens from the token endpoint. (You can even enable client credentials for apps that already have consumer keys and secrets.) For details on API key validation, see API keys

4.2.1.2 Implementing the client credentials grant type

With the client credentials grant type, an app sends it's own credentials (the Client ID and Client Secret) to a GenerateAccessToken endpoint on SAP API Management. If the credentials are valid, it returns an access token to the client app.

4.2.1.2.1 About this topic

This topic offers a general description of the OAuth 2.0 client credentials grant type and discusses how to implement this flow on SAP API Management.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

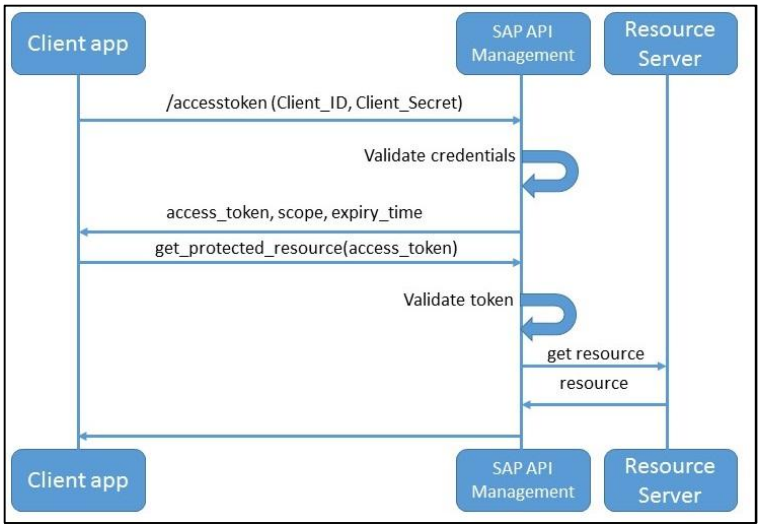
4.2.1.2.2 Use cases

Most typically, this grant type is used when the app is also the resource owner. For example, an app may need to access a backend cloud-based storage service to store and retrieve data that it uses to perform its work, rather than data specifically owned by the end user. This grant type flow occurs strictly between a client app and the authorization server. An end user does not participate in this grant type flow.

4.2.1.2.3 Roles

Roles specify the "actors" that participate in the OAuth flow. Let's do a quick overview of the client credentials roles to help illustrate where SAP API Management fits in. For a complete discussion of OAuth 2.0 roles, see the IETF OAuth 2.0 specification.

- Client App -- The app that needs access to the user's protected resources. Typically, with this flow, the app runs on server rather than locally on the user's laptop or device.
- SAP API Management -- In this flow, SAP API Management is the OAuth authorization server. It's role is to generate access tokens, validate access tokens, and pass authorized requests for protected resources on to the resource server.
- Resource Server -- The backend service that stores the protected data that the client app needs permission to access. If you are protecting API proxies hosted on SAP API Management, then SAP API Management is also the resource server.



CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.2.4 Steps in the client credentials flow

Here is a summary of the steps required to implement the client credentials code grant type where SAP API Management serves as the authorization server. Remember, with this flow, the client app simply presents its client ID and client secret, and if they are valid, SAP API Management returns an access token.

Prerequisite: The client app must be registered with SAP API Management to obtain the client ID and client secret keys. See Registering client apps for details.

1. Client requests an access token

To receive an access token, the client POSTs an API call to SAP API Management with the values for client ID and client secret obtained from a registered developer app. In addition, the parameter grant_type=client_credentials must be passed as a query parameter. (However, you can configure the OAuthV2 policy to accept this parameter in the request header or body).

For example:

```
$ curl -i -H 'ContentType: x-www-form-urlencoded' -X POST 'https://docs-
test.<host:port>/oauth/accesstoken' -d
'grant_type=client_credentials&client_id=ns4fQc14Zg4hKFCNaSzArVuwszX95X&client_secret=
ZIjFyTsNgQNyxI'
```

> **Note**
>
> Although you can pass the client_id and client_secret values as query parameters as shown above, it's a good practice to pass them as a base64 URL encoded string in the Authorization header. To do this, you need to use a base64 encoding tool or utility to encode the two values together with colon separating them. Like this: aBase64EncodeFunction(clientidvalue:clientsecret). So, the example above would be encoded like this:

result = aBase64EncodeFunction(ns4fQc14Zg4hKFCNaSzArVuwszX95X:ZIjFyTsNgQNyxI) / / Note the colon separating the two values.

The result of base64 encoding the above string is:
bnM0ZlFjMTRaZzRoS0ZDTmFTekFyVnV3c3pYOTVYOlpJakZ5VHNOZ1FOeXhJOg==

Then, make the token request like this:

```
$ curl -i -H 'ContentType: x-www-form-urlencoded' -X POST 'https://docs-
test.<host:port>/oauth/accesstoken' -d 'grant_type=client_credentials' -H
'Authorization: Basic
bnM0ZlFjMTRaZzRoS0ZDTmFTekFyVnV3c3pYOTVYOlpJakZ5VHNOZ1FOeXhJOg=='
```

2. SAP API Management validates the credentials

Note that the API call is sent to the /accesstoken endpoint. This endpoint has a policy attached to it that validates the app's credentials. That is, the policy compares the submitted keys with the ones that SAP API Management created when the app was registered.

3. SAP API Management returns a response

If the credentials are okay, SAP API Management returns an access token to the client. If not, an error is returned.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4. The client calls the protected API

Now, with a valid access token, the client can make calls to the protected API. In this scenario, requests are made to SAP API Management (the proxy), and it is responsible for validating the access token before passing the API call along to the target resource server.

4.2.1.2.5 Configuring flows and policies

As the authorization server, SAP API Management processes requests for access tokens. As the API developer, you need to create a proxy with a custom flow to handle token requests and add and configure an OAuthV2 policy. This section explains how to configure that endpoint.

Custom flow configuration

The easiest way to show how the API proxy flow is configured is to show the XML flow definition. Here's an example API proxy flow designed to process an access token request. For example, when a request comes in and the path suffix matches /accesstoken, the GenerateAccessToken policy is triggered.

```
<Flows>
  <Flow name="GetAccessToken">
        <!-- This policy flow is triggered when the URI path suffix
        matches /oauth/accesstoken. Publish this URL to app developers
        to use when obtaining an access token using an auth code
        -->
    <Condition>proxy.pathsuffix == "/oauth/accesstoken"</Condition>
    <Request>
      <Step><Name>GenerateAccessToken</Name></Step>
    </Request>
  </Flow>
</Flows>
```

Configure the flow with a policy

You need to attach a policy to the endpoint, as follows.

Get access token

This policy is attached to the /accesstoken path. It uses the OAuthV2 policy with the GenerateAccessToken operation specified.

```
<OAuthV2 name="GetAccessToken">
  <Operation>GenerateAccessToken</Operation>
  <ExpiresIn>3600000</ExpiresIn>
  <SupportedGrantTypes>
    <GrantType>client_credentials</GrantType>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  </SupportedGrantTypes>
  <GenerateResponse/>
</OAuthV2>
```

The API call to obtain the access token is a POST and includes an Authorization header with the base64 encoded client_id + client+secret and the query parameter grant_type=client_credentials. It can also include optional parameters for scope and state. For example:

```
$ curl -i -H 'ContentType: x-www-form-urlencoded' -X POST 'https://docs-
test.<host:port>/oauth/accesstoken' -d 'grant_type=client_credentials' -H
'Authorization: Basic
c3FIOG9vSGV4VHo4QzAySVgT1JvNnJoZ3ExaVNyQWw6WjRsanRKZG5lQk9qUE1BVQ'
```

4.2.1.2.6 Attaching the verify access token policy

To protect your API with OAuth 2.0 security, you need to add an OAuthV2 policy with the VerifyAccessToken operation. This policy checks that incoming requests have a valid access token. If the token is valid, SAP API Management processes the request. If it is not valid, SAP API Management returns an error.

```
<OAuthV2 async="false" continueOnError="false" enabled="true"
name="VerifyAccessToken">
    <DisplayName>VerifyAccessToken</DisplayName>
    <ExternalAuthorization>false</ExternalAuthorization>
    <Operation>VerifyAccessToken</Operation>
    <SupportedGrantTypes/>
    <GenerateResponse enabled="true"/>
    <Tokens/>
</OAuthV2>
```

4.2.1.2.7 Calling the protected API

To call an API that is protected with OAuth 2.0 security, you need to present a valid access token. The correct pattern is to include the token in an Authorization header, as follows: Note that the access token is also referred to as a "bearer token".

```
$ curl -H "Authorization: Bearer UAj2yiGAcMZGxfN2DhcUbl9v8WsR" \
  http://myorg-test.<host:port>/v0/weather/forecastrss?w=12797282
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.3 Implementing the authorization code grant type

Authorization code is one of the most commonly used OAuth 2.0 grant types. The authorization code flow is a "three-legged OAuth" configuration. In this configuration, the user authenticates himself with the resource server and gives the app consent to access his protected resources without divulging username/passwords to the client app.

4.2.1.3.1 About this topic

This topic offers a general description and overview of the OAuth 2.0 authorization grant type flow and discusses how to implement this flow on SAP API Management.

4.2.1.3.2 Use cases

This grant type is intended for apps that are written by third-party developers who do not have a trusted business relationship with the API provider. For example, developers who register for public API programs should not generally be trusted. With this grant type, the user's credentials on the resource server are never shared with the app.

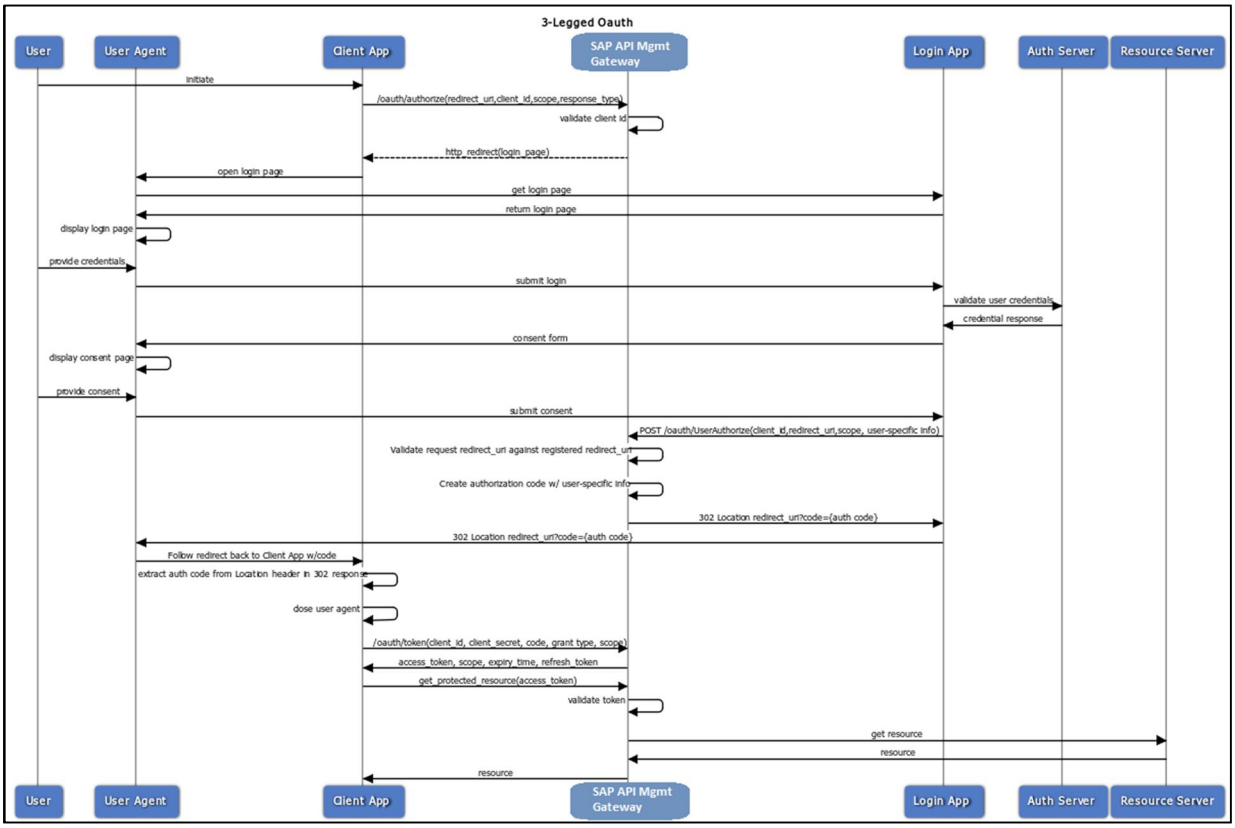
4.2.1.3.3 Code sample

You can find a complete, working sample implementation of the authorization code grant type on SAP API Management in the api-platform-samples repo on GitHub. See the oauth-advanced sample in the api-platform-samples/sample-proxies directory.

4.2.1.3.4 Flow diagram

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The following flow diagram illustrates the authorization code OAuth flow with SAP API Management serving as the authorization server.



4.2.1.3.5 Steps in the authorization code flow

Here is a summary of the steps required to implement the authorization code grant type where SAP API Management serves as the authorization server. Remember, the key to this flow is that the client never gets to see the user's credentials on the resource server.

Prerequisite: The client app must be registered with SAP API Management to obtain the client ID and client secret keys.

1. User initiates the flow

When the app needs to access the user's protected resources from a resource server (for example, contacts list on a social media site), it sends an API call to SAP API Management, which validates the client's ID and, if it's valid, redirects the user's browser to a login page where the user will enter her credentials. The API call includes information the client app obtained when it was registered: the client ID and redirect URI.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

2. User enters credentials

The user now sees a login page where she is asked to enter her login credentials. If the login is successful, we go to the next step.

3. User gives consent

In this step, the user gives the app consent to access her resources. The consent form typically includes scope selections, where the user can choose what the app is permitted to do on the resource server. For example, the user may give read-only permission, or permission for the app to update resources.

4. The login app sends a request SAP API Management

If the login and consent are successful, the login app POSTs data to the /authorizationcode endpoint of SAP API Management. The data includes the redirect URI, client ID, scope, any user-specific information it wishes to include, and an indication that the login was successful.

> **Note**
>
> It is important to note that the client app never sees this information. It is handled server-side and negotiated between the login app and SAP API Management.

5. SAP API Management generates an authorization code

When SAP API Management receives a GET request from the login app on its /authorizationcode endpoint, two things happen. First, it determines that the login was successful (by checking HTTP status or some other means). Next it checks to be sure that the redirect URI sent from the login app matches the redirect URI that was specified when the app was registered with SAP API Management. If everything is okay, it generates an authorization code. For example:

```
http://myorg-
test.<host:port>/oauth/authorizationcode?client_id={consumer_key}&response_type=code&r
edirect_uri={redirect_uri}&scope=scope1%20scope2&state={some_string}
```

6. SAP API Management sends the authorization code back to the client

SAP API Management sends a 302 redirect with the auth code attached as a query parameter to the client.

7. The client retrieves the authorization code and requests an access code from SAP API Management

Now with a valid auth code, the client can request an access token from SAP API Management. It does this by POSTing the client ID and client secret keys (obtained when the app was registered on SAP API Management), the grant type, and scope. By including the client ID and secret keys SAP API Management can verify that the client app is the one that was registered. For example:

```
$ curl https://{org_name}-
test.<host:port>/my_oauth_proxy/accesstoken?code=Xyz123&grant_type=authorization_code
-X POST -d 'client_id=bBGAQrXgivA9lKu7NMPyoYpKNhGar6K&client_secret=hAr4GngA9vAyvI4'
```

8. The client receives an access token

If everything is successful, SAP API Management returns an access token to the client. The access token will have an expiration, and it will be valid only for the scope specified by the user when she gave consent to the app to access her resources.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

9. The client calls the protected API

Now, with a valid access code, the client can make calls to the protected API. In this scenario, requests are made to SAP API Management (the proxy), and SAP API Management is responsible for validating the access token before passing the API call along to the target resource server. Access tokens are passed in an Authorization header. For example:

```
$ curl -H "Authorization: Bearer ylSkZIjbdWybfs4fUQe9BqP0LH5Z" http://{org-name}-
{env}.<host:port>/weather/forecastrss?w=12797282
```

4.2.1.3.6 Configuring flows and policies

As the authorization server, SAP API Management needs to process a number of OAuth requests: for access tokens, auth codes, refresh tokens, login page redirects, etc. There are two fundamental steps to configure these endpoints:

- Creating custom flows
- Adding and configuring OAuthV2 policies

4.2.1.3.7 Custom flow configuration

You typically configure this grant type flow so that each step or "leg" of the flow is defined by a flow in the SAP API Management proxy. Each flow has an endpoint and a policy that performs the OAuth-specific task required, such as generating an authorization code or an access token. For example, as shown in the XML below,the /oauth/authorizationcode endpoint has an associated policy called GenerateAuthCode (which is an OAuthV2 policy with the GenerateAuthorizationCode operation specified).

The easiest way to show the flow configuration is with an XML example. See the in-line comments for information about each flow. This is an example -- names of flows and paths can be configured however you wish.

```
<Flows>
  <Flow name="RedirectToLoginApp">
         <!--
         Publish this URI to developers to use for their 'login' link
         -->
    <Condition>proxy.pathsuffix == "/oauth/authorize"</Condition>
    <Request>
       <Step><Name>RedirectToLoginPage</Name></Step>
    </Request>
  </Flow>
  <Flow name="GetAuthCode">
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        <!--
        Call this URL from your Login app after you authenticate the user.
        The policy will automatically return the auth code in the response to the
        redirect_uri registered by the calling app
        -->
    <Condition>proxy.pathsuffix == "/oauth/authorizationcode"</Condition>
    <Request>
      <Step><Name>GenerateAuthCode</Name></Step>
    </Request>
  </Flow>
  <Flow name="GetAccessToken">
        <!-- This policy flow is triggered when the URI path suffix
        matches /oauth/accesstoken. Publish this URL to app developers
        to use when obtaining an access token using an auth code
        -->
    <Condition>proxy.pathsuffix == "/oauth/accesstoken"</Condition>
    <Request>
      <Step><Name>GenerateAccessToken</Name></Step>
    </Request>
  </Flow>
</Flows>
```

4.2.1.3.8 Configure the flows with policies

Each endpoint has a policy associated with it. Let's see examples of the policies.

Login redirect

This is the /oauth/authorize path. The attached policy is responsible for redirecting the user to a login app, where the end user can safely authenticate and authorize the client app to access his protected resources without divulging his username and password to the client app. You can accomplish this with a service callout policy, JavaScript, Node.js, or other means.

The API call to do the request is a GET and requires the query parameters client_id, response_type, redirect_uri, scope, and state.

```
$ curl http://myorg-
test.<host:port>/oauth/authorize?client_id={consumer_key}&response_type=code&redirect_
uri={redirect_uri}&scope=scope1%20scope2&state={some_string}
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The login app typically POSTs a redirect response back. SAP API Management then directs the user's browser to the registered callback URL. It's a best practice not to send the response directly back to the user agent/client to avoid exposing sensitive information.

Get auth code

This is the /oauth/authorizationcode path. It uses the OAuthV2 policy with the GenerateAuthorizationCode operation specified.

```
<OAuthV2 async="false" continueOnError="false" enabled="true" name="GetAuthCode">
    <DisplayName>GetAuthCode</DisplayName>
    <Operation>GenerateAuthorizationCode</Operation>
    <ExpiresIn>600000</ExpiresIn>
    <GenerateResponse/>
</OAuthV2>
```

The API call to obtain the authorization code is a GET and requires the query parameters client_id, response_type, and optionally scope and state, as shown in this example:

```
$curl http://myorg-
test.<host:port>/oauth/authorizationcode?client_id={consumer_key}&response_type=code&s
cope=scope1%20scope2&state={some_string}
```

Get access token

This policy is attached to the /oauth/accesstoken path. It uses the OAuthV2 policy with the GenerateAccessCode operation specified. In this case, the grant_type parameter is expected as a query param:

```
<OAuthV2 name="GetAccessToken">
  <Operation>GenerateAccessToken</Operation>
  <ExpiresIn>360000000</ExpiresIn>
  <SupportedGrantTypes>
  <GrantType>authorization_code</GrantType>
    </SupportedGrantTypes>
  <GrantType>request.queryparam.grant_type</GrantType>
  <GenerateResponse/>
</OAuthV2>
```

The API call to obtain the access code is a POST and must include the client_id, client_secret, grant_type=code, and, optionally, scope. For example:

```
$ curl https://{org_name}-test.<host:port>/oauth/accesstoken?grant_type=code -X POST -
d 'client_id=bBGAQrXgivA9lKu7NMPyoYpVKNhGar6K&client_secret=hAr4Gn0gA9vAyvI4'
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

This is just a basic summary. A production example includes many other policies for building up URLs, doing transformations, and performing other tasks.

4.2.1.3.9 Attaching the verify access token policy

Attach a VerifyAccessToken policy (OAuthV2 policy with the VerifyAccessToken operation specified) to the beginning of any flow that accesses a protected API, so that it is executed whenever a request for a protected resources comes in. SAP API Management checks to be sure each request has a valid access token. If not, an error is returned.

```
<OAuthV2 async="false" continueOnError="false" enabled="true"
name="VerifyAccessToken">
    <DisplayName>VerifyAccessToken</DisplayName>
    <ExternalAuthorization>false</ExternalAuthorization>
    <Operation>VerifyAccessToken</Operation>
    <SupportedGrantTypes/>
    <GenerateResponse enabled="true"/>
    <Tokens/>
</OAuthV2>
```

4.2.1.3.10 Calling the protected API

To call an API that is protected with OAuth 2.0 security, you need to present a valid access token. The correct pattern is to include the token in an Authorization header, as follows: Note that the access token is also referred to as a "bearer token".

```
$ curl -H "Authorization: Bearer UAj2yiGAcMZGxfN2DhcUbl9v8WsR" \
  http://myorg-test.<host:port>/v0/weather/forecastrss?w=12797282
```

4.2.1.4 Implementing the password grant type

The resource owner password (or "password") grant type is mostly used in cases where the app is highly trusted. In this configuration, the user provides his resource server credentials (username/password) to the client app, which sends them in an access token request to SAP API Management. An identity server validates the credentials, and if they are valid, SAP API Management proceeds to mint an access token and returns it to the app.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.4.1 About this topic

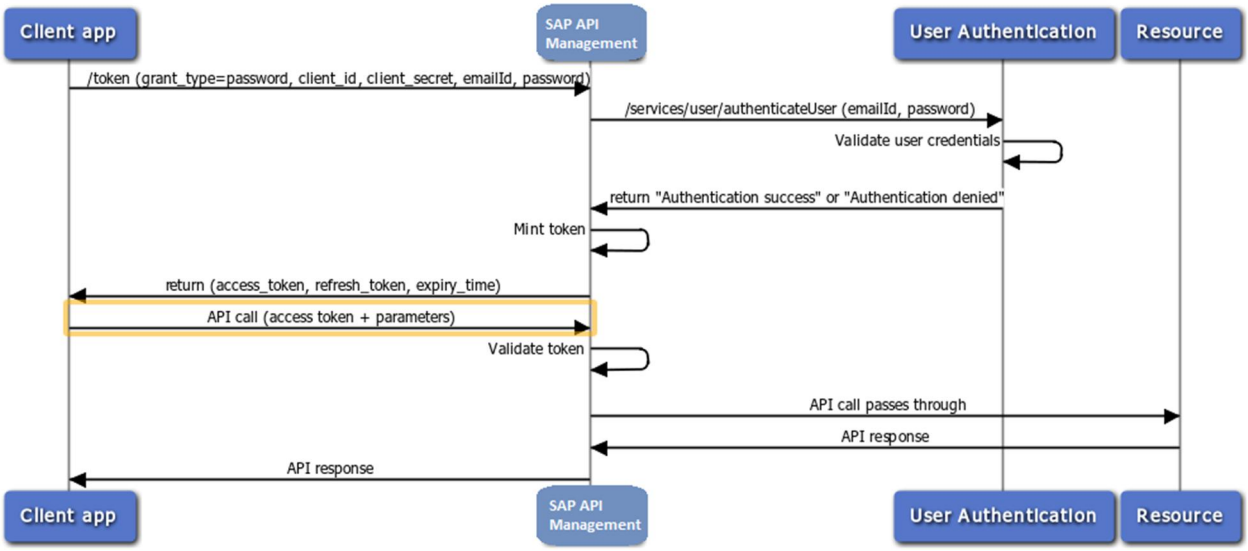
This topic offers a general description and overview of the OAuth 2.0 resource owner password grant type flow and discusses how to implement this flow on SAP API Managment.

4.2.1.4.2 Use cases

This grant type is intended for highly trusted or privileged apps because the user is required to give his resource server credentials to the app. Typically, the app provides a login screen where the user enters her credentials.

4.2.1.4.3 Flow diagram

The following flow diagram illustrates the resource owner password grant type flow with SAP API Management serving as the authorization server.



4.2.1.4.4 Steps in the password grant type flow

Here is a summary of the steps required to implement the password grant type where SAP API Management serves as the authorization server.

Prerequisite: The client app must be registered with SAP API Management to obtain the client ID and client secret keys.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

1. User initiates the flow and enters credentials

When the app needs to access the user's protected resources (for example, the user selects a button in the app), the user is redirected to a login form.

2. App requests an access token from SAP API Management

The app sends an access token request, including the user's credentials, to a GenerateAccessToken endpoint on SAP API Management.

Here is a sample POST request, which includes the required parameters for this grant type:

```
$ curl -i -H 'ContentType: x-www-form-urlencoded' -X POST 'https://docs-
test.<host:port>/oauth/token -d 'grant_type=password&username=the-user-
name&password=the-users-password' -H 'Authorization: Basic
c3FIOG9vSGV4VHo4QzAySVg5T1JvNnJoZ3ExaVNyQWw6WjRsanRKZG5lQk9qUE1BVQ'
```

3. SAP API Management receives the request and processes the login credentials

A typical pattern is to use an ExtractVariables policy to extract the username and password from the request and save them as flow variables. Then, use a Service Callout or JavaScript policy to call the identity service, sending in the credentials. This could be an LDAP service or any service that you wish to use to validate the credentials.

If the identity service validates the credentials, and returns a 200 response, then SAP API Management will continue processing the request; otherwise, it stops processing and returns an error to the client app.

4. The OAuthV2 policy executes

If the credentials are valid, the next processing step is to execute an OAuthV2 policy configured for the password grant type. Here is an example. The <UserName> and <PassWord> elements are required, and you can retrieve them from the flow variables that were saved with the ExtractVariables policy.

```
<OAuthV2 name="GetAccessToken">
  <Operation>GenerateAccessToken</Operation>
  <ExpiresIn>360000000</ExpiresIn>
  <SupportedGrantTypes>
  <GrantType>password</GrantType>
    </SupportedGrantTypes>
  <GrantType>request.queryparam.grant_type</GrantType>
  <UserName>login</UserName>
  <PassWord>password</PassWord>
  <GenerateResponse/>
</OAuthV2>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

If this policy succeeds, a response is generated back to the client containing an access token. The response is in JSON format. Here's an example. Note that access_token is one of the elements:

```
{
    "issued_at": "1420258685042",
    "scope": "READ",
    "application_name": "ce1e94a2-9c3e-42fa-a2c6-1ee01815476b",
    "refresh_token_issued_at": "1420258685042",
    "status": "approved",
    "refresh_token_status": "approved",
    "api_product_list": "[PremiumWeatherAPI]",
    "expires_in": "1799",
    "developer.email": "tesla@weathersample.com",
    "organization_id": "0",
    "token_type": "BearerToken",
    "refresh_token": "IFl7jlijYuexu6XVSSjLMJq8SVXGOAAq",
    "client_id": "5jUAdGv9pBouF0wOH5keAVI35GBtx3dT",
    "access_token": "I6daIgMSiUgYX1K2qgQWPi37ztS6",
    "organization_name": "docs",
    "refresh_token_expires_in": "0",
    "refresh_count": "0"
}
```

5. The client calls the protected API

Now, with a valid access code, the client can make calls to the protected API. In this scenario, requests are made to SAP API Management (the proxy), and SAP API Management is responsible for validating the access token before passing the API call along to the target resource server. Access tokens are passed in an Authorization header. For example:

```
$ curl -H "Authorization: Bearer I6daIgMSiUgYX1K2qgQWPi37ztS6
" http://{org-name}-{env}.<host:port>/weather/forecastrss?w=12797282
```

4.2.1.5 OAuth 2.0: Configuring a new API proxy

For convenience, all organizations on SAP API Management come preconfigured with a set of OAuth 2.0 endpoints that implement the client credentials grant type. This topic explains how to protect an API using this default configuration.

The client credentials grant type defines a procedure for issuing access tokens in exchange for app credentials. These app credentials are the consumer key and secret pair that SAP API Management issues for each app that is registered in an organization.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

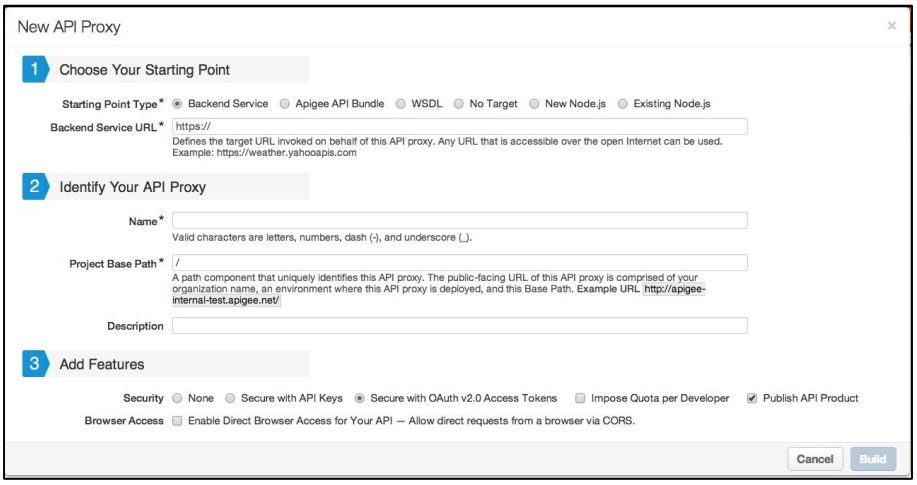
4.2.1.5.1 Add OAuth 2.0 to a new API proxy

You can easily add OAuth verification to an API when you create a new API proxy.

1. Log in to your SAP API Management account.
2. Select APIs > API Proxies.
3. Choose + API Proxy.
4. Fill out the New API Proxy dialog.
5. Select the radio button next to Secure with OAuth v2.0 Access Tokens.

When you select this option, two policies will be attached to the newly created API proxy, one to verify access tokens and another to strip the access token after it has been verified.

Note that the Publish API Product checkbox becomes selectable and is automatically selected. Check this if you want to automatically generate a product when you build the new API proxy. The autogenerated product will be created with an association to the new API proxy. If you have an existing product with which you want to associate this new API, be sure to clear this checkbox so that you don't create an unnecessary product.



4.2.1.5.2 Working with the default OAuth configuration

Each organization (even a free trial org) on SAP API Management is provisioned with an OAuth token endpoint. The endpoint is preconfigured with policies in the API proxy called oauth. You can begin using the token endpoint as soon as you create an account on SAP API Management.

The default OAuth endpoint exposes the following endpoint URI:

```
/oauth/client_credential/accesstoken
```

Publish this URI to developers who need to obtain access tokens. App developers configure their apps to call this endpoint, presenting their consumer key and secret pairs to obtain access tokens.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The default client credentials token endpoint is exposed over the network at the following URL:

```
https://{org_name}-{env_name}.<host:port>/oauth/client_credential/accesstoken
```

For example, if your organization name is "apimakers", the URL would be:

```
https://apimakers-test.<host:port>/oauth/client_credential/accesstoken
```

This is the URL that developers call to obtain access tokens.

> Tip
>
> By default, out-of-the-box OAuth endpoints are only deployed in the test environment. Before they are available in prod, you must explicitly deploy the API proxy called oauth to prod.

4.2.1.6 Registering client apps

To participate in OAuth 2.0 flows on SAP API Management, client apps must be registered.

4.2.1.6.1 What is registration?

Registration allows SAP API Management (the authorization server) to uniquely identify your app. When you register your app, you receive back two keys: a client ID and client secret. The app needs these keys when negotiating for access tokens with the authorization server.

4.2.1.6.2 Quick steps

For development and testing, you can use one of the pre-registered developer apps to obtain keys.

If you want to register a new app:

1. Log in to your SAP API Management account.
2. From the main menu, select Publish > Developer apps.
3. Choose + Developer App.
4. Fill out the form:
    1. Enter a name.
    2. Select a developer (you can choose one of the default developers or create your own).
    3. (Optional) Enter a callback URL. This is used for "three-legged" OAuth grant type flows. This is where SAP API Management redirects the user after she completes authentication (login) with the resource server. It

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

has to be a complete URL, so you might enter something likehttps://www.example.com. For more about three-legged OAuth, see Implementing the authorization code grant type.

4. Add a product. You can select one of the default products (or any product you wish). Or create your own.
5. Ignore custom attributes for now.
6. Choose Save.

5. Find your new app in the list of developer apps and select it.
6. Now, you can choose Show to see the Consumer ID (client ID) and Consumer Secret (client secret) values.

4.2.1.7 Obtaining client credentials

This topic shows you how to obtain client credentials (also called developer keys) for development and testing purposes using an out-of-the-box developer app and product. For production situations, the steps are similar, however you would use your own developer apps, products, and other entities.

4.2.1.7.1 What are client credentials?

To participate in any OAuth 2.0 flow, all client apps must be registered with SAP API Management (the authorization server). When you register your app, you will be assigned two keys: the Consumer ID and Consumer Secret. The Consumer ID is a public key and Consumer Secret must never be made public. These client credential keys allow SAP API Management to uniquely identify the client app.

Terminology: The IETF OAuth 2.0 specification refers to client credentials as the client identifier and client secret. The SAP API Management management UI refers to them as the Consumer ID and the Consumer Secret. These terms are synonymous.

4.2.1.7.2 Quick steps

These quick steps show you how to obtain developer keys for one of the out-of-the-box apps that are configured when you created your SAP API Management organization. You can use these keys mainly for development and testing:

1. Log in to your SAP API Management account.
2. Select Publish > Developer Apps to open the Developer Apps overview page.
3. Choose Weather App to open the Weather App overview page. Notice that the developer associated with the app is Nicolai Tesla. This app and developer were provisioned by default when your organization was created. All developer apps must have a developer associated with them.
4. In the Products section, next to the Premium Weather API product, choose Show to expose the Consumer ID and Consumer Secret values.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

5. Copy and save those two values -- you will need to use them later to make API calls to obtain access tokens.

Tip

It's important to be sure that the product associated with the keys is enabled for the same environment where the API you are securing is deployed. If not, you will receive this error when the access token is validated: Invalid API call as no apiproduct match found.

4.2.1.7.3 Getting client credentials with SAP API Management APIs

You can also obtain the consumer key and secret for an app by calling the management API. First, get the list of apps in your organization by making the following API call:

```
$ curl https://<host:port>/v1/o/{org_name}/apps \
-u myname:mypass
```

This call returns a list of apps by app ID.

```
[ "da496fae-2a04-4a5c-b2d0-709278a6f9db", "50e3e831-175b-4a05-8fb6-05a54701af6e" ]
```

You can retrieve an app's profile by making a simple GET call on the app ID:

```
$ curl https://<host:port>/v1/o/{org_name}/apps/{app_id} \
-u myname:mypass
```

For example:

```
$ curl https://<host:port>/v1/o/{org_name}/apps/da496fae-2a04-4a5c-b2d0-709278a6f9db \
-u myname:mypass
```

The API call returns the profile of the app you specified. For example, an app profile for weatherapp has the following JSON representation:

```
{
  "accessType" : "read",
  "apiProducts" : [ ],
  "appFamily" : "default",
  "appId" : "da496fae-2a04-4a5c-b2d0-70928a6f9db",
  "attributes" : [ ],
  "callbackUrl" : "http://weatherapp.com",
  "createdAt" : 1380290158713,
  "createdBy" : "noreply_admin@sap.com",
  "credentials" : [ {
    "apiProducts" : [ {
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
      "apiproduct" : "PremiumWeatherAPI",
      "status" : "approved"
    } ],
    "attributes" : [ ],
    "consumerKey" : "bBGAQrXgivA9lKu7NMPyYpVKNhGar6K",
    "consumerSecret" : "hAr4Gn0gA9vyvI4",
    "expiresAt" : -1,
    "issuedAt" : 1380290161417,
    "scopes" : [ ],
    "status" : "approved"
  } ],
  "developerId" : "5w95xGkpnjzDBT4",
  "lastModifiedAt" : 1380290158713,
  "lastModifiedBy" : "noreply_admin@sap.com",
  "name" : "weatherapp",
  "scopes" : [ ],
  "status" : "approved"
}
```

Note the values for **consumerKey** and **consumerSecret**.

4.2.1.8 Understanding OAuth endpoints

As the authorization server, SAP API Management needs to have appropriate OAuth endpoints set up so that clients can request authorization codes and access tokens. This topic offers a quick introduction to endpoints.

4.2.1.8.1 What is an OAuth endpoint?

An OAuth endpoint is a URL that is exposed by SAP API Management in your organization. OAuth defines token endpoints, authorization endpoints, and refresh endpoints. Apps call these endpoints to get access tokens, to refresh access tokens, and, in some cases, to get authorization codes. These endpoints refer to specific OAuth 2.0 policies that execute when the endpoint is called.

Here's an example. In this flow, the GenerateAccessToken policy is executed when the proxy path matches /token.

```
  <Flow name="generate-access-token">
          <Description/>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
            <Request>
                <Step>
                    <FaultRules/>
                    <Name>GenerateAccessToken</Name>
                </Step>
            </Request>
            <Response/>
            <Condition>(proxy.pathsuffix MatchesPath &quot;/token&quot;) and
(request.verb = &quot;POST&quot;)</Condition>
        </Flow>
```

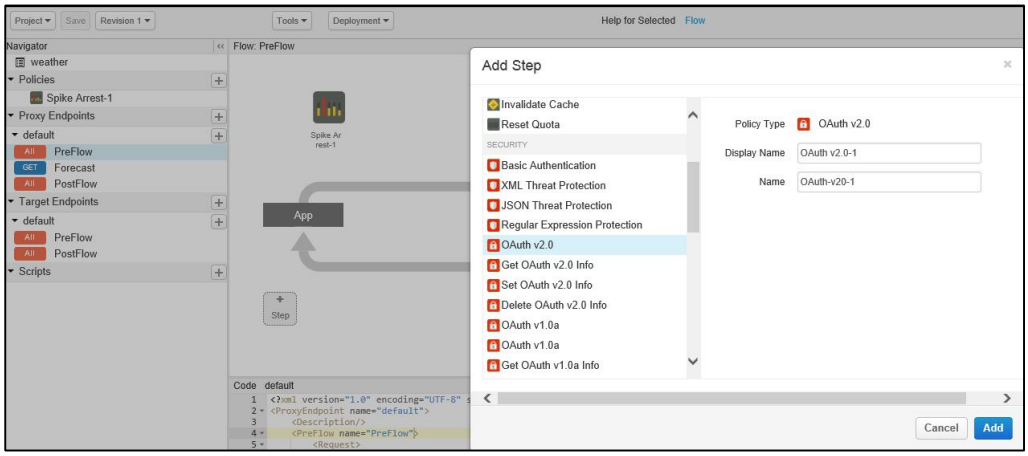
Here's an example API call to the /token endpoint on SAP API Management.

```
$ curl -i -H 'ContentType: x-www-form-urlencoded' -X POST 'https://docs-
test.<host:port>/oauth/token' -d 'grant_type=client_credentials' -H 'Authorization:
Basic c3FIOG9vSGV4VHo4QzAySVg5T1JvNnJoZ3ExaVNyQWw6WjRsanRKZG5lQk9qUE1BVQ'
```

4.2.1.8.2 Using the default endpoints

The quickest way to see how endpoints are set up is to examine the default "oauth" proxy. This proxy is installed for you when you create a new SAP API Management organization. It sets up OAuth endpoints that support the client credentials grant type. Let's take a look.

1. Log in to your SAP API Management account.
2. Select APIs > API Proxies from the main menu.
3. In the list of proxies, select the one called oauth.
4. In the proxy overview page, select the Develop tab to bring up the proxy editor.



You'll see in this view the policies and flows that are configured to support this OAuth grant type flow.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.8.3 Best practice: Create your own OAuth proxy

The default oauth proxy is only supports the client credentials grant type, and is mainly provisioned to support examples. For your OAuth 2.0 implementation, it's a common practice to create your own OAuth endpoint proxy where you define your specific set of conditional flows and attach OAuthV2 policies.

The OAuth proxy that you create does not make any backend calls. Instead, the OAuth proxy acts as a standalone service. Once you have set up the conditional flows and attached the policies, app developers can call the URLs exposed by your API proxy to get access tokens, refresh access tokens, and, in the case of the authorization code grant type, authorization codes.

4.2.1.9 Requesting access tokens and authorization codes

In this topic, we show you how to request access tokens and authorization codes, configure OAuth 2.0 endpoints, and configure policies for each supported grant type.

> **Note**
>
> These examples show the most basic configurations possible. In particular, the OAuthV2 policy includes many optional configurable elements that are not shown in this topic.

4.2.1.9.1 Requesting an access token: authorization code grant type

This section explains how to request an access token using the authorization code grant type flow. For an introduction to OAuth 2.0 grant types, see Introduction to OAuth 2.0.

Sample request

```
 $ curl -i -H 'ContentType: x-www-form-urlencoded' -X POST 'https://docs-
test.<host:port>/oauth/accesstoken' -d
'code=I9dMGHAN&grant_type=authorization_code&redirect_uri=http://example-callback.com'
-H 'Authorization: Basic
c3FIOG9vSGV4VHo4QzAyg5T1JvNnJoZ3ExaVNyQWw6WjRsanRKZG5lQk9qUE1BVQ'
```

Required parameters

By default, these parameters must be `x-www-form-urlencoded` and specified in the request body (as shown in the sample above); however, it is possible to change this default by configuring the `<GrantType>`, `<Code>`, and `<RedirectUri>` elements in the OAuthV2 policy that is attached to this /accesstoken endpoint.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- grant_type - Must be set to the value `authorization_code`.
- code - The authorization code received from the /authorize endpoint (or whatever you choose to name it). To request an access token in the authorization code grant type flow, you must first obtain an authorization code.
- redirect_uri - You must provide this parameter if the redirect_uri parameter was included in the prior authorization code request. If the `redirect_uri` parameter was not included in the authorization code request, and if you do not provide this parameter, then this policy uses the value of the Callback URL that was provided when the developer app was registered.

Optional parameters

- state - A string that will be sent back with the response. Typically used to prevent cross-site request forgery attacks.
- scope - Allows you to filter the list of API products with which the minted token can be used.

Authentication

You must pass the Client ID and Client Secret either as a Basic Authentication header (Base64-encoded) or as form parameters client_id andclient_secret. You obtain these values from a registered developer app.

Sample endpoint

Here's a sample endpoint configuration for generating an access token. It'll execute the GenerateAccessToken policy, which must be configured to support the authorization_code grant type.

```
...
       <Flow name="generate-access-token">
            <Description/>
            <Request>
                <Step>
                    <FaultRules/>
                    <Name>GenerateAccessToken</Name>
                </Step>
            </Request>
            <Response/>
            <Condition>(proxy.pathsuffix MatchesPath &quot;/token&quot;) and
(request.verb = &quot;POST&quot;)</Condition>
        </Flow>
...
```

Sample policy

This is a basic GenerateAccessToken policy that is configured to accept the authorization_code grant type.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<OAuthV2 async="false" continueOnError="false" enabled="true"
name="GenerateAccessToken">
    <DisplayName>GenerateAccessToken</DisplayName>
    <Operation>GenerateAccessToken</Operation>
    <SupportedGrantTypes>
      <GrantType>authorization_code</GrantType>
    </SupportedGrantTypes>
    <GenerateResponse enabled="true"/>
</OAuthV2>
```

Returns

With <GenerateResponse> enabled, the policy returns a JSON response that includes the access token, as shown below. Note that theauthorization_code grant type supports minting both access and refresh tokens. For example:

```
{
    "issued_at": "1420262924658",
    "scope": "READ",
    "application_name": "ce1e94a2-9c3e-42fa-a2c6-1ee01815476b",
    "refresh_token_issued_at": "1420262924658",
    "status": "approved",
    "refresh_token_status": "approved",
    "api_product_list": "[PremiumWeatherAPI]",
    "expires_in": "1799",
    "developer.email": "tesla@weathersample.com",
    "organization_id": "0",
    "token_type": "BearerToken",
    "refresh_token": "fYACGW7OCPtCNDEnRSnqFlEgogboFPMm",
    "client_id": "5jUAdGv9pBouF0wOH5keAVI35GBtx3dT",
    "access_token": "2l4IQtZXbn5WBJdL6EF7uenOWRsi",
    "organization_name": "docs",
    "refresh_token_expires_in": "0",
    "refresh_count": "0"
}
```

If <GenerateResponse> is set to false, the policy does not return a response. Instead, it populates the following set of flow variables with data pertaining to the access token grant.

```
oauthv2accesstoken.{policy-name}.access_token
oauthv2accesstoken.{policy-name}.expires_in
oauthv2accesstoken.{policy-name}.refresh_token
oauthv2accesstoken.{policy-name}.refresh_token_expires_in
oauthv2accesstoken.{policy-name}.refresh_token_issued_at
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
oauthv2accesstoken.{policy-name}.refresh_token_status
```

For example:

```
oauthv2accesstoken.GenerateAccessToken.access_token
oauthv2accesstoken.GenerateAccessToken.expires_in
oauthv2accesstoken.GenerateAccessToken.refresh_token
oauthv2accesstoken.GenerateAccessToken.refresh_token_expires_in
oauthv2accesstoken.GenerateAccessToken.refresh_token_issued_at
oauthv2accesstoken.GenerateAccessToken.refresh_token_status
```

4.2.1.9.2 Requesting an access token: client credentials grant type

This section explains how to request an access token using the client credentials grant type flow.

Sample request

```
$ curl -i -H 'ContentType: x-www-form-urlencoded' -X POST 'https://docs-test.<host:port>/oauth/accesstoken' -d 'grant_type=client_credentials' -H 'Authorization: Basic c3FIOG9vSGV4VHoAyg5T1JvNnJoZ3ExaVNyQWw6WjRsanRKZG5lQk9qUE1BVQ'
```

Required parameters

By default, the required grant_type parameter must be x-www-form-urlencoded and specified in the request body (as shown in the sample above); however, it is possible to change this default by configuring the <GrantType> element in the OAuthV2 policy that is attached to this /accesstokenendpoint. For example, you could elect to pass the parameter in a query parameter.

- grant_type - Must be set to the value client_credentials.

Optional parameters

- state - A string that will be sent back with the response. Typically used to prevent cross-site request forgery attacks.
- scope - Allows you to filter the list of API products with which the minted token can be used.

Authentication

You must pass the Client ID and Client Secret either as a Basic Authentication header (Base64-encoded) or as form parameters client_id andclient_secret. You obtain these values from the registered developer app associated with the request.

Sample endpoint

Here's a sample endpoint configuration for generating an access token. It'll execute the GenerateAccessToken policy, which must be configured to support the client_credentials grant type.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
...
       <Flow name="generate-access-token">
            <Description/>
            <Request>
                <Step>
                    <FaultRules/>
                    <Name>GenerateAccessToken</Name>
                </Step>
            </Request>
            <Response/>
            <Condition>(proxy.pathsuffix MatchesPath &quot;/token&quot;) and
(request.verb = &quot;POST&quot;)</Condition>
        </Flow>
...
```

Sample policy

This is a basic GenerateAccessToken policy that is configured to accept the client_credentials grant type.

```
<OAuthV2 async="false" continueOnError="false" enabled="true"
name="GenerateAccessToken">
    <DisplayName>GenerateAccessToken</DisplayName>
    <Operation>GenerateAccessToken</Operation>
    <SupportedGrantTypes>
      <GrantType>client_credentials</GrantType>
    </SupportedGrantTypes>
    <GenerateResponse enabled="true"/>
</OAuthV2>
```

Returns

With <GenerateResponse> enabled, the policy returns a JSON response. Note that with the client_credentials grant type, refresh tokens are not supported. Only an access token is minted. For example:

```
{
    "issued_at": "1420260525643",
    "application_name": "ce1e94a2-9c3e-42fa-a2c6-1ee01815476b",
    "scope": "READ",
    "status": "approved",
    "api_product_list": "[PremiumWeatherAPI]",
    "expires_in": "1799",
    "developer.email": "tesla@weathersample.com",
    "organization_id": "0",
    "token_type": "BearerToken",
    "client_id": "5jUAdGv9pBouF0wOH5keAVI35GBtx3dT",
    "access_token": "XkhU2DFnMGIVL2hvsRHLM00hRWav",
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    "organization_name": "docs",
    "refresh_token_expires_in": "0",
    "refresh_count": "0"
}
```

If <GenerateResponse> is set to false, the policy does not return a response. Instead, it populates the following set of flow variables with data pertaining to the access token grant.

```
oauthv2accesstoken.{policy-name}.access_token
oauthv2accesstoken.{policy-name}.expires_in
```

For example:

```
oauthv2accesstoken.GenerateAccessToken.access_token
oauthv2accesstoken.GenerateAccessToken.expires_in
```

4.2.1.9.3 Requesting an access token: password grant type

This section explains how to request an access token using the resource owner password credentials (password) grant type flow.

Sample request

```
$ curl -i -H 'ContentType: x-www-form-urlencoded' -X POST 'https://docs-
test.<host:port>/oauth/token -d 'grant_type=password&username=the-user-
name&password=the-users-password' -H 'Authorization: Basic
c3FIOG9vSGV4VHo4QzAySVg5T1JvNnJoZ3ExaVNyQWw6WjRsanRKZG5lQk9qUE1BVQ'
```

Required parameters

By default, these parameters must be x-www-form-urlencoded and specified in the request body (as shown in the sample above); however, it is possible to change this default by configuring the <GrantType>, <Username>, and <Password> elements in the OAuthV2 policy that is attached to this /token endpoint.

User credentials are typically validated against a credential store using an LDAP or JavaScript policy.

- grant_type - Must be set to the value password.
- username - The resource owner's user name.
- password - The resource owner's password.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Optional parameters

- state - A string that will be sent back with the response. Typically used to prevent cross-site request forgery attacks.
- scope - Allows you to filter the list of API products with which the minted token can be used.

Authentication

You must pass the Client ID and Client Secret either as a Basic Authentication header (Base64-encoded) or as form parameters client_id andclient_secret. You obtain these values from the registered developer app associated with the request.

Sample endpoint

Here's a sample endpoint configuration for generating an access token. It'll execute the GenerateAccessToken policy, which must be configured to support the password grant type.

```
...
       <Flow name="generate-access-token">
            <Description/>
            <Request>
                <Step>
                    <FaultRules/>
                    <Name>GenerateAccessToken</Name>
                </Step>
            </Request>
            <Response/>
            <Condition>(proxy.pathsuffix MatchesPath &quot;/token&quot;) and
(request.verb = &quot;POST&quot;)</Condition>
        </Flow>
...
```

Sample policy

This is a basic GenerateAccessToken policy that is configured to accept the password grant type.

```
<OAuthV2 async="false" continueOnError="false" enabled="true"
name="GenerateAccessToken">
    <DisplayName>GenerateAccessToken</DisplayName>
    <Operation>GenerateAccessToken</Operation>
    <SupportedGrantTypes>
      <GrantType>password</GrantType>
    </SupportedGrantTypes>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <GenerateResponse enabled="true"/>
</OAuthV2>
```

Returns

With <GenerateResponse> enabled, the policy returns a JSON response. Note that with the password grant type, both an access token and refresh token are minted. For example:

```
{
    "issued_at": "1420258685042",
    "scope": "READ",
    "application_name": "ce1e94a2-9c3e-42fa-a2c6-1ee01815476b",
    "refresh_token_issued_at": "1420258685042",
    "status": "approved",
    "refresh_token_status": "approved",
    "api_product_list": "[PremiumWeatherAPI]",
    "expires_in": "1799",
    "developer.email": "tesla@weathersample.com",
    "organization_id": "0",
    "token_type": "BearerToken",
    "refresh_token": "IFl7jlijYuexu6XVSSjLMJq8SVXGOAAq",
    "client_id": "5jUAdGv9pBouF0wOH5keAVI35GBtx3dT",
    "access_token": "I6daIgMSiUgYX1K2qgQWPi37ztS6",
    "organization_name": "docs",
    "refresh_token_expires_in": "0",
    "refresh_count": "0"
}
```

If <GenerateResponse> is set to false, the policy does not return a response. Instead, it populates the following set of flow variables with data pertaining to the access token grant.

```
oauthv2accesstoken.{policy-name}.access_token
oauthv2accesstoken.{policy-name}.expires_in
oauthv2accesstoken.{policy-name}.refresh_token
oauthv2accesstoken.{policy-name}.refresh_token_expires_in
oauthv2accesstoken.{policy-name}.refresh_token_issued_at
oauthv2accesstoken.{policy-name}.refresh_token_status
```

For example:

```
oauthv2accesstoken.GenerateAccessToken.access_token
oauthv2accesstoken.GenerateAccessToken.expires_in
oauthv2accesstoken.GenerateAccessToken.refresh_token
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
oauthv2accesstoken.GenerateAccessToken.refresh_token_expires_in
oauthv2accesstoken.GenerateAccessToken.refresh_token_issued_at
oauthv2accesstoken.GenerateAccessToken.refresh_token_status
```

4.2.1.9.4 Requesting an access token: implicit grant type

This section explains how to request an access token using the implicit grant type flow.

Sample request

```
curl -X POST -H 'ContentType: x-www-form-urlencoded' 'https://docs-
test.<host:port>/oauth/implicit?response_type=token&client_id=ABC123&redirect_uri=http
://callback-example.com'
```

Required parameters

By default, these parameters must be query parameters (as shown in the sample above); however, it is possible to change this default by configuring the <ResponseType>, <ClientId>, and <Redirect_Uri> elements in the OAuthV2 policy that is attached to this /token endpoint.

User credentials are typically validated against a credential store using an LDAP service callout or JavaScript policy.

- response_type - Must be set to the value token.
- client_id - The client ID of a registered developer app.
- redirect_uri - This parameter is mandatory if a Callback URI was not provided when the client developer app was registered. If a Callback URL was provided at client registration, it will be compared to this value and must match exactly.

Optional parameters

- state - A string that will be sent back with the response. Typically used to prevent cross-site request forgery attacks.
- scope - Allows you to filter the list of API products with which the minted token can be used.

Authentication

You must pass the Client ID and Client Secret either as a Basic Authentication header (Base64-encoded) or as form parameters client_id and client_secret. You obtain these values from the registered developer app associated with the request.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Sample endpoint

Here's a sample endpoint configuration for generating an access token. It'll execute the GenerateAccessTokenImplicitGrant policy.

```
...
       <Flow name="generate-access-token-implicit">
            <Description/>
            <Request>
                <Step>
                    <FaultRules/>
                    <Name>GenerateAccessTokenImplicitGrant</Name>
                </Step>
            </Request>
            <Response/>
            <Condition>(proxy.pathsuffix MatchesPath &quot;/implicit&quot;) and
(request.verb = &quot;POST&quot;)</Condition>
        </Flow>
...
```

Sample policy

This is a basic GenerateAccessTokenImplicitGrant policy that processes token requests for the implicit grant type flow.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OAuthV2 async="false" continueOnError="false" enabled="true"
name="GenerateAccessTokenImplicit">
    <DisplayName>GenerateAccessTokenImplicit</DisplayName>
    <Operation>GenerateAccessTokenImplicitGrant</Operation>
    <GenerateResponse enabled="true"/>
</OAuthV2>
```

Returns

With <GenerateResponse> enabled, the policy returns a 302 Location redirect in the response header. The redirect points to the URL specified in theredirect_uri parameter and is appended with the access token and token expiration time. Note that the implicit grant type does not support refresh tokens. For example:

```
https://callback-example.com#expires_in=1799&access_token=In4dKm4ueoGZRbIYJhC9yZCmTFw5
```

If <GenerateResponse> is set to false, the policy does not return a response. Instead, it populates the following set of flow variables with data pertaining to the access token grant.

```
oauthv2accesstoken.{policy-name}.access_token
oauthv2accesstoken.{policy-name}.expires_in
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

For example:

```
oauthv2accesstoken.GenerateAccessToken.access_token
oauthv2accesstoken.GenerateAccessToken.expires_in
```

Requesting an authorization code

If you're using the authorization code grant type flow, you need to obtain an authorization code before you can request an access token.

Sample request

```
curl -X POST -H 'ContentType: x-www-form-urlencoded' 'http://myorg-
test.<host:port>/oauth/authorize?client_id={consumer_key}&response_type=code'
```

where an OAuthV2 GenerateAuthorizationCode policy is attached at the /oauth/authorize proxy endpoint (see the sample endpoint below).

Required parameters

By default, these parameters must be query parameters (as shown in the sample above); however, it is possible to change this default by configuring the <ResponseType>, <ClientId>, and <Redirect_Uri> elements in the OAuthV2 policy that is attached to this /authorize endpoint.

- response_type - Must be set to the value code.
- client_id - The client ID of a registered developer app.

Optional parameters

- redirect_uri - If a full (not partial) Callback URI is specified in the registered client app, this parameter is optional; otherwise, it is required. The callback is the URL where SAP API Management sends the newly minted auth code.
- state - A string that will be sent back with the response. Typically used to prevent cross-site request forgery attacks.
- scope - Allows you to filter the list of API products with which the minted token can be used.

Authentication

Does not require basic authentication, however the client ID of the registered client app must be supplied in the request.

Sample endpoint

Here's a sample endpoint configuration for generating an authorization code:

```
...
      <Flow name="generate-auth-code">
          <Description/>
          <Request>
              <Step>
                  <FaultRules/>
                  <Name>GenerateAuthorizationCode</Name>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
                </Step>
            </Request>
            <Response/>
            <Condition>(proxy.pathsuffix MatchesPath &quot;/authorize&quot;) and
(request.verb = &quot;POST&quot;)</Condition>
        </Flow>
...
```

Sample policy

This is a basic GenerateAuthorizationCode policy.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OAuthV2 async="false" continueOnError="false" enabled="true"
name="GenerateAuthorizationCode">
    <DisplayName>GenerateAuthCode</DisplayName>
    <Operation>GenerateAuthorizationCode</Operation>
    <GenerateResponse enabled="true"/>
</OAuthV2>
```

Returns

With <GenerateResponse> enabled, the policy returns ?code query parameter to the redirect_uri (Callback URI) location with the authorization code attached. It is sent via a 302 browser redirect with the URL in the Location header of the response. For example: ?code=123456.

If <GenerateResponse> is set to false, the policy does not return a response. Instead, it populates the following set of flow variables with data pertaining to the authorization code.

```
oauthv2authcode.{policy-name}.code
oauthv2authcode.{policy-name}.scope
oauthv2authcode.{policy-name}.redirect_uri
oauthv2authcode.{policy-name}.client_id
```

For example:

```
oauthv2authcode.GenerateAuthorizationCode.code
oauthv2authcode.GenerateAuthorizationCode.scope
oauthv2authcode.GenerateAuthorizationCode.redirect_uri
oauthv2authcode.GenerateAuthorizationCode.client_id
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.9.5 Refreshing an access token

A refresh token is a credential you use to obtain an access token, typically after the access token has expired or becomes invalid. A refresh token is returned in the response when you receive an access token.

To request a new access token using a refresh token:

Sample request

```
curl -X POST 'https://myorg-test.<host:port>/my_oauth_endpoint/refresh_accesstoken' -d
'grant_type=refresh_token&refresh_token=my-refresh-token' -H "Content-type:
application/x-www-form-urlencoded"
```

Required parameters

By default, these parameters must be x-www-form-urlencoded and specified in the request body (as shown in the example above); however, it is possible to change this default for the grant type by configuring the <GrantType> element in the OAuthV2 policy that is attached to this/accesstoken endpoint.

User credentials are typically validated against a credential store using an LDAP or JavaScript policy.

- grant_type - Must be set to the value refresh_token.
- refresh_token - The refresh token associated with the access token you wish to renew.

Optional parameters

- state - A string that will be sent back with the response. Typically used to prevent cross-site request forgery attacks.
- scope - Allows you to filter the list of API products with which the minted token can be used.

Authentication

You must pass the Client ID and Client Secret either as a Basic Authentication header (Base64-encoded) or as form parameters client_id andclient_secret.

Here's a sample endpoint configuration for generating an access token using a refresh token. It'll execute the RefreshAccessToken policy.

```
 ...
       <Flow name="generate-refresh-token">
            <Description/>
            <Request>
                <Step>
                    <FaultRules/>
                    <Name>RefreshAccessToken</Name>
                </Step>
            </Request>
            <Response/>
            <Condition>(proxy.pathsuffix MatchesPath &quot;/refresh&quot;) and
(request.verb = &quot;POST&quot;)</Condition>
       </Flow>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
...
```

Sample policy

This is a basic RefreshAccessToken policy that is configured to accept the refresh_token grant type.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OAuthV2 async="false" continueOnError="false" enabled="true"
name="RefreshAccessToken">
    <DisplayName>RefreshAccessToken</DisplayName>
    <ExternalAuthorization>false</ExternalAuthorization>
    <Operation>RefreshAccessToken</Operation>
    <GenerateResponse enabled="true"/>
</OAuthV2>
```

Returns

With <GenerateResponse> enabled, the policy returns a JSON response containing the new access token. Note that the refresh_accesstokengrant type supports minting both access and new refresh tokens. For example:

```
{
    "issued_at": "1420301470489",
    "application_name": "ce1e94a2-9c3e-42fa-a2c6-1ee01815476b",
    "scope": "READ",
    "refresh_token_issued_at": "1420301470489",
    "status": "approved",
    "refresh_token_status": "approved",
    "api_product_list": "[PremiumWeatherAPI]",
    "expires_in": "1799",
    "developer.email": "tesla@weathersample.com",
    "token_type": "BearerToken",
    "refresh_token": "8fKDHLryAD9KFBsrpixlq3qPJnG2fdZ5",
    "client_id": "5jUAdGv9pBouF0wOH5keAVI35GBtx3dT",
    "old_access_token_life_time": "285832",
    "access_token": "jmZ2Hqv3iNsABUtAAsfWR3QGNctw",
    "organization_name": "docs",
    "refresh_token_expires_in": "0",
    "refresh_count": "2"
}
```

If <GenerateResponse> is set to false, the policy does not return a response. Instead, it populates the following set of flow variables with data pertaining to the access token grant.

```
oauthv2accesstoken.{policy-name}.access_token
oauthv2accesstoken.{policy-name}.expires_in
oauthv2accesstoken.{policy-name}.refresh_token
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
oauthv2accesstoken.{policy-name}.refresh_token_expires_in
oauthv2accesstoken.{policy-name}.refresh_token_issued_at
oauthv2accesstoken.{policy-name}.refresh_token_status
```

For example:

```
oauthv2accesstoken.GenerateAccessToken.access_token
oauthv2accesstoken.GenerateAccessToken.expires_in
oauthv2accesstoken.GenerateAccessToken.refresh_token
oauthv2accesstoken.GenerateAccessToken.refresh_token_expires_in
oauthv2accesstoken.GenerateAccessToken.refresh_token_issued_at
oauthv2accesstoken.GenerateAccessToken.refresh_token_status
```

4.2.1.9.6 Encoding basic authentication credentials

When you make an API call to request a token or auth code, it's a good practice to pass the client_id and client_secret values as a base64 URL encoded string in the Authorization header. To do this, you need to use a base64 encoding tool or utility to encode the two values together with colon separating them.

For example:

```
result = aBase64EncodeFunction(ns4fQc14Zg4hKFCNaSzArVuwszX95X:ZIjFyTsNgQNyxI)
```

Note the colon separating the two values.

where ns4fQc14Zg4hKFCNaSzArVuwszX95X is the client_id and ZIjFyTsNgQNyxI is the client secret.

The result of base64 encoding the above string is: bnM0ZlFjMTRaZzRoS0ZDTmFTekFyVnV3c3pYOTVYOlpJakZ5VHNOZ1FOeXhJOg==

Then, make the token request as follows:

```
$ curl -i -H 'ContentType: x-www-form-urlencoded' -X POST 'https://docs-
test.<host:port>/oauth/accesstoken' -d 'grant_type=client_credentials' -H
'Authorization: Basic
bnM0ZlFjMTRaZzRoS0ZDTmFTekFyVnV3c3pYOTVYOlpJakZ5VHNOZ1FOeXhJOg=='
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.10 Customizing tokens and auth codes

4.2.1.10.1 About customizing tokens and auth codes

SAP API Management generates and distributes OAuth access tokens, refresh tokens, and authorization codes to apps. SAP API Management stores those tokens and codes and uses them to authorize consumer apps.

When it generates these OAuth artifacts, it also generates a profile that contains metadata related to the token or code. For example, the default access token profile contains name/value pairs that define expiration time, the associated app and developer, and so on.

The JSON representation of an SAP API Management access token looks like the following:

```
{
  "issued_at" : "1372170159093",
  "application_name" : "ccd1803b-b557-4520-bd62-ddd3abf8e501",
  "scope" : "READ",
  "status" : "approved",
  "api_product_list" : "[FreeProduct]",
  "expires_in" : "3599",
  "developer.email" : "joe@weathersample.com",
  "organization_id" : "0",
  "refresh_token" : "82XMXgDyHTpFyXOaApj8C2AGIPnN2IZe",
  "client_id" : "deAVedE0W9Z9U35PAMaAJYphBJCGdrND",
  "access_token" : "shTUmeI1geSKin0TODcGLXBNe9vp",
  "organization_name" : "apifactory",
  "refresh_count" : "0"
}
```

Sometimes its useful to add custom attributes to an access token. For example, you might wish to embed a department name, a customer ID, or, more technically, a session identifier. You can also get and set these custom attributes at runtime using policies (as explained below).

4.2.1.10.2 Customizing tokens with the OAuthV2 policy

You can configure custom token attributes directly in the OAuthV2 policy that generates the tokens or auth codes.

```
<Attributes>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  <Attribute name="attr_name1" ref="flow.variable"
display="true|false">value1</Attribute>
  <Attribute name="attr_name2" ref="flow.variable"
display="true|false">value2</Attribute>
</Attributes>
```

Where display is set to true, custom attributes are returned in the response, where they may be viewable by the app end user.

Where display is set to false, custom attributes are stored in the data store, but are not returned in the response message.

For example, to add the User-Agent associated with the request to an access token:

```
<OAuthV2 name="GenerateAccessToken">
  <Operation>GenerateAccessToken
  </Operation>
  <ExpiresIn>1000</ExpiresIn>
  <GenerateResponse />
  <SupportedGrantTypes>
    <GrantType>client_credentials</GrantType>
  </SupportedGrantTypes>
  <GrantType>request.queryparam.grant_type</GrantType>
  <Attributes>
    <Attribute name="user-agent" ref="request.header.user-agent"
display="false"></Attribute>
  </Attributes>
</OAuthV2>
```

4.2.1.10.3 Getting and setting custom attributes at runtime

In some situations, you will need to update the profile of an access token at runtime while an API call is being processed on SAP API Management. To help with this, SAP API Management provides policies for getting and setting token attributes. Use these policies when you need tokens to be updated at runtime, such as at the time when the token or code is generated by it.

The AccessToken element value is set to the name of the variable where the access token can be located: either in the request message, or in some other variable where it might be populated by an `ExtractVariables` policy.

You can also use SAP API Management APIs to manage a token's profile.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.11 Approving and revoking access tokens

4.2.1.11.1 Revoking access and refresh tokens

In some cases, apps are required to explicitly revoke or invalidate tokens, for example, when a user logs out of an OAuth-enabled app. If you revoke a token, it can be re-approved anytime before it expires.

You can also revoke/approve client IDs associated with products and developer apps.

The procedure for token revocation is defined by the OAuth 2.0 Token Revocation specification.

SAP API Management provides an InvalidateToken operation that enables you to configure a dedicated token revocation endpoint. By publishing the URI of this endpoint, you enable app developers to invalidate tokens issued by it.

Token expiration is independent of the state of the token (approved or revoked). If a refresh token has expired, you cannot use it to refresh an access token.

Here's an example configuration for the OAuthV2 policy and the InvalidateToken operation. In this case, both the access token and its associated refresh token are revoked. Technically, they are both revoked because the cascade flag is set to true. For more information about how the cascade flag works, see the Token element's attributes section below.

```
<OAuthV2 name="InvalidateToken">
  <Operation>InvalidateToken</Operation>
  <Tokens>
    <Token type="accesstoken" cascade="true">flow.variable</Token>
  </Tokens>
</OAuthV2>
```

4.2.1.11.2 <Tokens>/<Token> element

Identifies the flow variable that specifies the token to be revoked. If developers are expected to submit a revocation request using a query parameter named `access_token`, for example, the correct flow variable will be: `request.queryparam.access_token`. To require the token in an HTTP header, for example, set this value to `request.header.access_token`.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.11.3 Attributes

- `type` (required, string): The token type identified by the variable specified. Supported values are accesstoken and refreshtoken:
  - To revoke an access token, specify type accesstoken.
  - To revoke both the access and refresh tokens, specify type refreshtoken. When it sees type refreshtoken, SAP API Management assumes the token is a refresh token. If that refresh token is found, then it is revoked. If that refresh token is not found, then SAP API Management checks to see if it is an access token. If the access token exists, then it is revoked.

    Note: If you pass an already invalidated token to an InvalidateToken policy, the policy doesn't return an error, although you might expect it to. Such an operation has no effect.
- `cascade` (optional, boolean, default: true) The primary use of this attribute is to revoke a refresh token without revoking its associated access token. Consider these cases:
  - Revoke a refresh token only and do not revoke its associated access token. To do this, set the <Token> type to refreshtoken and set cascade to false.
  - Revoke both the access token and the refresh token. To do this, set the <Token> type to accesstoken. The value of cascade can be eithertrue (the default) or false. If you set it to true, then both the access token and the refresh token are revoked. If you set it to false, the access token is revoked, and the refresh token is unusable. See the Note below for more explanation.
  - Revoke an access token and do not revoke its associated refresh token. Not supported. See the Note below for more explanation.

Note: For security reasons, it is not possible to use a refresh token to refresh an access token that has been revoked. A refresh token can only be used to refresh an access token that has expired. Therefore, you cannot use the cascade attribute to revoke only an access token. For example, if you set the <Token> type to accesstoken, and set cascade=false, the access token is revoked (as expected); however, the associated refresh token is unusable. It cannot be used to refresh the revoked access token. The primary use case for the cascade attribute is when you want to only revoke a refresh token. In that case, set the <Token> type to refreshtoken, and set cascade=false. The refresh token will be revoked, but it's associated access token will remain valid (until it expires or is revoked).

4.2.1.11.4 Approving access and refresh tokens

Use the ValidateToken operation to "re-approve" a revoked token. That is, when you apply this operation, the status of the targeted access or refresh token is changed from 'revoked' to 'approved'. You can validate any revoked token that has not already expired.

```
<OAuthV2 name="ValidateToken">
  <Operation>ValidateToken</Operation>
  <Tokens>
    <Token type="refreshtoken" cascade="true">flow.variable</Token>
  </Tokens>
</OAuthV2>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

<Tokens> / <Token> element

Identifies the flow variable that specifies the token to be validated. If developers are expected to submit a validation request using a query parameter named access_token, for example, the correct flow variable will be: request.queryparam.access_token. To require the token in an HTTP header, for example, set this value to request.header.access_token.

Attributes

- `type` (required, string) The token type identified by the variable specified. Supported values are accesstoken and refreshtoken.

4.2.1.12 Enable retrieval and revocation of OAuth 2.0 access tokens by end user ID, app id, or both

This section describes how to enable retrieval and revocation of OAuth 2.0 access tokens by end user ID, app ID, or both.

4.2.1.12.1 About OAuth access tokens

App IDs are automatically added to an OAuth access token. Therefore, after you enable token access for an organization as described below, you can access tokens by app ID.

To retrieve and revoke OAuth 2.0 access tokens by end user ID, an end user ID must be present in the access tokens. The procedure below describes how add an end user ID to an existing token.

By default, when SAP API Management generates an OAuth 2.0 access token, the token has the format shown below:

```
{
 "issued_at" : "1421847736581",
 "application_name" : "a68d01f8-b15c-4be3-b800-ceae8c456f5a",
 "scope" : "READ",
 "status" : "approved",
 "api_product_list" : "[PremiumWeatherAPI]",
 "expires_in" : "3599",
 "developer.email" : "tesla@weathersample.com",
 "organization_id" : "0",
 "token_type" : "BearerToken",
 "client_id" : "k3nJyFJIA3p62DWOkLO6OJNi87GYXFmP",
 "access_token" : "7S22UqXGJDTuUADGzJzjXzXSaGJL",
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
 "organization_name" : "myorg",
 "refresh_token_expires_in" : "0",
 "refresh_count" : "0"
}
```

Note the following:

- The `application_name` field contains the UUID of the app associated with the token. If you enable retrieval and revocation of OAuth 2.0 access tokens by app ID, then this is the app ID you use.
- The `access_token` field contains the OAuth 2.0 access token value.

There is no field for end user ID in the default OAuth access token. To enable retrieval and revocation of OAuth 2.0 access tokens by end user ID, you have to configure the OAuth 2.0 policy to include the user ID in the token, as described in the procedure below. Note that if you only want to retrieve and revoke OAuth 2.0 access tokens by app ID, then there is no need to enable access by end user ID.

The end user ID is the string that SAP API Management uses as the developer ID, not the developer's email address. You can determine the developer's ID from the developer's email address by using Get Developer. After you configure SAP API Management to include the end user ID in the token, it is included as the app_enduserfield, as shown below:

```
{
 "issued_at" : "1421847736581",
 "application_name" : "a68d01f8-b15c-4be3-b800-ceae8c456f5a",
 "scope" : "READ",
 "app_enduser" : "6ZG094fgnjNf02EK",
 "status" : "approved",
 "api_product_list" : "[PremiumWeatherAPI]",
 "expires_in" : "3599",
 "developer.email" : "tesla@weathersample.com",
 "organization_id" : "0",
 "token_type" : "BearerToken",
 "client_id" : "k3nJyFJIA3p62DWOkLO6OJNi87GYXFmP",
 "access_token" : "7S22UqXGJDTuUADGzJzjXzXSaGJL",
 "organization_name" : "myorg",
 "refresh_token_expires_in" : "0",
 "refresh_count" : "0"
}
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.12.2 Enabling access to OAuth 2.0 tokens by user ID and app ID

Refer to the operations guide for this information.

4.2.1.13 Revoking and approving developer app keys

This topic explains how to use the UI and APIs to revoke or approve API keys for products in developer apps and for developer apps.

> **Note**
>
> If you need to revoke an access token, you can do that directly by calling a properly configured OAuthV2 endpoint.

4.2.1.13.1 Using the UI to revoke the key for a specific API product in a developer app

You can revoke the key associated with a specific API product through the management UI. The effect is that the API resources defined in that API product will no longer be accessible unless the key is re-approved.

1. Log in to your SAP API Management account.
2. From the main menu, select Publish > Developer apps.
3. From the list of apps, select the one that contains the key you wish to revoke.
4. In the Developer app page, choose Edit.
5. In the Products section, choose Revoke to revoke the key for the product you wish to revoke.

Tip: You can use the Get Developer App Details API to get the JSON representation of a developer app to discover the approval status of products in that app. In the following JSON representation of a developer app, you can see where the API Product called "Weather-Product" has been revoked:

```
{
  "accessType": "",
  "appFamily": "default",
  "appId": "6ed3a4d1-4733-439a-80a4-0d71149ec9ad",
  "attributes": [
    {
      "name": "DisplayName",
      "value": "AnotherTestApp"
    },
    {
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
      "name": "Notes",
      "value": ""
    }
  ],
  "callbackUrl": "",
  "createdAt": 1415728893126,
  "createdBy": "wwitman@sap.com",
  "credentials": [
    {
      "apiProducts": [
        {
          "apiproduct": "Weather-Product",
          "status": "revoked"
        }
      ],
      "attributes": [],
      "consumerKey": "giIC9Au6XP82wJ1oxZuQU4L75OdNKLhb",
      "consumerSecret": "SXg8JizqeTA8j3gX",
      "expiresAt": -1,
      "issuedAt": 1415728893154,
      "scopes": [],
      "status": "approved"
    }
  ],
  "developerId": "Z2S37rxX2Suzkwtg",
  "lastModifiedAt": 1420682967414,
  "lastModifiedBy": "wwitman@sap.com",
  "name": "AnotherTestApp",
  "scopes": [],
  "status": "approved"
}
```


4.2.1.13.2 Using the UI to approve the key for a specific API product in a developer app

You can approve a previously revoked consumer key for an API product in a developer app through the management UI. The client app will once again be able to access the APIs in that product.

1. Log in to your SAP API Management account.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

2. From the main menu, select Publish > Developer apps.
3. From the list of apps, select the one that contains the key you wish to revoke.
4. In the Developer app page, choose Edit.
5. In the Products section, choose Approve to reinstate the key.

4.2.1.13.3 Revoking and approving keys for API products with the SAP API Management API

SAP API Management provides APIs that let you manage API key status, depending on your use case. In the developer app JSON below, the location of each approve/revoke flag is indicated for each of these three cases:

(1) Revoking/approving keys for specific API products in a developer app. This API does exactly what the UI operations described previously in this topic do

(2) Revoking/approving specific key for a developer app. Revoking the key renders it unusable for the app to use it to access an API. Any access tokens associated with a revoked app key will remain active, but SAP API Management checks the status of the app key first. If the status is set to "revoked," SAP API Management will not allow the call go through.

(3) Revoke/approve the API key status of an entire developer app. A revoked app cannot access any API products and cannot invoke any API managed by SAP API Management.


```
{
  "accessType": "",
  "appFamily": "default",
  "appId": "6ed3a4d1-4733-439a-80a4-0d71149ec9ad",
  "attributes": [
    {
      "name": "DisplayName",
      "value": "AnotherTestApp"
    },
    {
      "name": "Notes",
      "value": ""
    }
  ],
  "callbackUrl": "",
  "createdAt": 1415728893126,
  "createdBy": "wwitman@sap.com",
  "credentials": [
    {
      "apiProducts": [
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        {
          "apiproduct": "Weather-Product",
          "status": "revoked" // (1)
        }
      ],
      "attributes": [],
      "consumerKey": "giIC9Au6XP82wJ1oxZuQU4L75OdNKLhb",
      "consumerSecret": "SXg8JizqeTA8j3gX",
      "expiresAt": -1,
      "issuedAt": 1415728893154,
      "scopes": [],
      "status": "approved" // (2)
    }
  ],
  "developerId": "Z2S37rxX2Suzkwtg",
  "lastModifiedAt": 1420682967414,
  "lastModifiedBy": "wwitman@sap.com",
  "name": "AnotherTestApp",
  "scopes": [],
  "status": "approved" // (3)
}
```

4.2.1.14 Sending an access token

As an app developer, you need to include an access token in any request to SAP API Management for a protected resource (an API that is protected with a VerifyAccessToken policy). Note that access tokens are also called "bearer tokens."

4.2.1.14.1 Sending an access token in a request

When you put a VerifyAccessToken policy at the front of your API proxy flow, apps must present a valid access token (also called a "bearer token") to consume your API. To do this, the app sends the access token in the request as an "Authorization" HTTP header.

For example:

```
$ curl -H "Authorization: Bearer ylSkZIjbdWybfs4fUQe9BqP0LH5Z" http://{org-name}-{env}.<host:port>/weather/forecastrss?w=12797282
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

SAP API Management will verify that the access token presented is valid, and then grant access to the API, returning the response to the app that made the request.

4.2.1.15 Verifying access tokens

When you call an API proxy on SAP API Management that has OAuth security, it is responsible for validating access tokens. Think of it as the gatekeeper -- no API call can pass through that does not have a valid access token.

4.2.1.15.1 Adding a VerifyAccessToken policy

To configure token validation, place an OAuthV2 policy with the VerifyAccessToken operation at the very beginning of the API proxy flow (the beginning of the ProxyEndpoint Preflow). If placed there, access tokens will be verified before any other processing takes place, and if a token is rejected, SAP API Management stops processing and returns an error back to the client.

1. Log in to your SAP API Management account.
2. From the main menu, select APIs > API Proxies.
3. From the list, select the proxy you wish to protect.
4. In the overview page, choose DEVELOP.
5. From the New Policy menu, select OAuth v2.0.
6. Fill out the policy form. Be sure the Attach Policy is checked, the flow is PreFlow, Proxy Endpoint default, and the segment is Request.
7. Choose Add.

The default policy is already configured with the VerifyAccessToken operation, so you do not have to do anything further:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OAuthV2 async="false" continueOnError="false" enabled="true" name="OAuth-v20-1">
    <DisplayName>OAuth v2.0 1</DisplayName>
    <FaultRules/>
    <Properties/>
    <Attributes/>
    <ExternalAuthorization>false</ExternalAuthorization>
    <Operation>VerifyAccessToken</Operation>
    <SupportedGrantTypes/>
    <GenerateResponse enabled="true"/>
    <Tokens/>
</OAuthV2>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Note

By default, VerifyAccessToken expects the access token to be sent in an Authorization header as a Bearer token. For example:

```
-H "Authorization: Bearer Rft3dqrs56Blirls56a"
```

4.2.1.16 Working with scopes

This topic discusses how to use OAuth 2.0 scopes on SAP API Management.

4.2.1.16.1 What is scope?

OAuth 2.0 scopes provide a way to limit the amount of access that is granted to an access token. For example, an access token issued to a client app may be granted READ and WRITE access to protected resources, or just READ access. You can implement your APIs to enforce any scope or combination of scopes you wish. So, if a client receives a token that has READ scope, and it tries to call an API endpoint that requires WRITE access, the call will fail.

In this topic, we'll discuss how scopes are assigned to access tokens and how SAP API Management enforces OAuth 2.0 scopes. After reading this topic, you'll be able to use scopes with confidence.

4.2.1.16.2 How are scopes assigned to access tokens?

When SAP API Management generates an access token, it may assign a scope to that token. To understand how this happens, you must first be familiar with these SAP API Management entities: API products, developers, and developer apps. We recommend that you review this material if you need to before continuing.

An access token is a long string of random-looking characters that allows it to verify incoming API requests (think of it as a stand-in for typical username/password credentials). Technically, the token is a key that refers to a collection of metadata that that looks like this:

```
{
  "issued_at" : "1416962591727",
  "application_name" : "0d3e1d41-a59f-4d74-957e-d4e3275d4781",
  "scope" : "A",
  "status" : "approved",
  "api_product_list" : "[scopecheck1-bs0cSuqS9y]",
  "expires_in" : "1799",
  "developer.email" : "scopecheck1-AdBmANhsag@sap.com",
  "organization_id" : "0",
  "token_type" : "BearerToken",
  "client_id" : "eTtB7w5lvk3DnOZNGReBlvGvIAeAywun",
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  "access_token" : "ODm47ris5AlEty8TDc1itwYPe5MW",
  "organization_name" : "wwitman",
  "refresh_token_expires_in" : "0",
  "refresh_count" : "0"
}
```

The token's metadata includes the actual access token string, expiry information, identification of the developer app, developer, and products associated with the token. You'll also notice that the metadata also includes "scope".

How does the token get its scope?

The first key to understanding scope is to remember that each product in a developer app can have zero or more scopes assigned to it. These scopes can be assigned when the product is created, or they can be added later. They exist as a list of names and are included in the "metadata" associated with each product.

You can name your scopes anything you wish. In simple cases, it is fine to use simple names like READ, WRITE, or DELETE. In more complex scenarios where there are multiple API products, each with multiple resources which each support multiple distinct actions, a WRITE on one resource is not equivalent to a WRITE on another resource. In these cases, it's a best practice to assign each scope a unique name, in the form of an URN. Examples of URNs include: https://www.examplecompany.com/private_catalog.readonly andurn:examplecompany:product_price:update.

Learn more about URNs in RFC3986.

When you create a developer app and add products to it, SAP API Management looks at all of the products in the developer app and creates a list of all of the scopes for those products (the app's master or global scope list -- a union of all recognized scopes).

> **Note**
>
> There is a case where the master list is overridden. It is possible to create a developer app using the SAP API Management API. Using the API, you can specify OAuth scopes for the app. App-specific scopes override the master list of scopes taken from the products that are included in the app. Note that the SAP API Management UI does not let you specify app-specific scopes. It is only possible if you use the API to create an app.

When a client app requests an access token from SAP API Management, it can optionally specify which scopes it would like to have associated with that token. For example, the following request asks for the scope "A". That is, the client is asking that the authorization server (SAP API Management) generate an access token that has scope "A" (giving the app authorization to call APIs that have scope "A"). The app sends a POST request like this:

```
curl -i -X POST -H Authorization: Basic Mg12YTk2UkEIyIBCrtro1QpIG -H content-
type:application/x-www-form-urlencoded http://myorg-
test.<host:port>/oauth/token?grant_type=client_credentials&scope=A
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

What happens?

When SAP API Management receives this request it knows which app is making the request and it knows which developer app the client registered (the client ID and client secret keys are encoded in the basic auth header). Because the scope query parameter is included, SAP API Management needs to decide if any of the API products associated with the developer app have scope "A". If they do, then an access token is generated with scope "A". Another way to look at this is that the scope query parameter is a kind of filter. If the developer app recognizes scopes "A, B, X", and the query parameter specifies "scope=X Y Z", then only scope "X" will be assigned to the token.

> **Note**
>
> If the value of the scope parameter matches none of the recognized scopes, then a token is still generated; however, that token will not be granted any scopes (it's scope metadata will be empty).

What if the client does not attach a scope parameter? In this case, SAP API Management generates a token that includes all of the scopes recognized by the developer app. It's important to understand that the default behavior is to return an access token that contains the union of all scopes for all of the products included in the developer app.

Let's say a developer app recognizes these scopes: A B C D. This is the app's master list of scopes. It could be that one product in the app has scope A and B, and a second one has scope C and D, or any combination. If the client does not specify a scope parameter (or if it specifies scope parameter with no value) the token will be granted all four scopes: A, B, C, and D. Again, the token receives a set of scopes that is the union of all the scopes recognized by the developer app.

There is one more case where the default behavior is to return an access token with all of the recognized scopes, and that is when the GenerateAccessToken policy (the SAP API Management policy that generates access tokens) does not specify a <Scope> element. For example, here's a GenerateAccessToken policy where <Scope> is specified. If that <Scope> element is missing (or if it is present but empty), then the default behavior is executed.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OAuthV2 async="false" continueOnError="false" enabled="true" name="OAuthV2-
GenerateAccessToken">
    <DisplayName>OAuthV2 - Generate Access Token</DisplayName>
    <Attributes>
      <Attribute name='hello' ref='system.time' display='false'>value1</Attribute>
    </Attributes>
    <Scope>A</Scope> <!-- Optional, space-separated list -->
    <GrantType>request.formparam.grant_type</GrantType>
    <ExternalAuthorization>false</ExternalAuthorization>
    <Operation>GenerateAccessToken</Operation>
    <SupportedGrantTypes>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
      <GrantType>client_credentials</GrantType>
    </SupportedGrantTypes>
  <GenerateResponse enabled="true"/>
</OAuthV2>
```

4.2.1.16.3 How are scopes enforced?

First, remember that on SAP API Management, access tokens are validated with the OAuthV2 policy (typically placed at the very beginning of a proxy flow). The policy must have the VerifyAccessToken operation specified. Let's look at this policy:

```
<OAuthV2 async="false" continueOnError="false" enabled="true" name="OAuthV2-
VerifyAccessTokenA">
    <DisplayName>Verify OAuth v2.0 Access Token</DisplayName>
    <ExternalAuthorization>false</ExternalAuthorization>
    <Operation>VerifyAccessToken</Operation>
    <Scope>A</Scope> <!-- Optional: space-separated list of scope names. -->
    <GenerateResponse enabled="true"/>
</OAuthV2>
```

Note the <Scope> element. It is used to specify which scopes the policy will accept.

> **Note**
>
> If more than one scope is specified (for example, <Scope>A B C</Scope>), then the policy will succeed if the access token includes any one of those scopes (like a logical 'OR' evaluation). If you want to enforce an 'AND' type of operation, where multiple scopes on a token are enforced, you can do that by creating multiple VeryifyAccessToken policies, each with a single distinct scope.

In this example, the policy will succeed only if the access token includes scope "A". If this <Scope> element is omitted or if it has no value, then the policy ignores the scope of the access token.

Now, with the ability to validate access tokens based on scope, you can design your APIs to enforce specific scopes. You do this by designing custom flows with scope-aware VerifyAccessToken policies attached to them.

Let's say your API has a flow defined for the endpoint /resourceA:

```
<Flow name="resourceA">
            <Condition>(proxy.pathsuffix MatchesPath "/resourceA") and (request.verb =
"GET")</Condition>
            <Description>Get a resource A</Description>
            <Request>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
            <Step>
                <Name>OAuthV2-VerifyAccessTokenA</Name>
            </Step>
        </Request>
        <Response>
            <Step>
                <Name>AssignMessage-CreateResponse</Name>
            </Step>
        </Response>
    </Flow>
```

When this flow is triggered (a request comes in with /resourceA in the path suffix), the OAuthV2-VerifyAccessTokenA policy is called immediately. This policy verifies that the access token is valid and it looks to see what scope(s) the token supports. If the policy is configured as the example below, with <Scope>A</Scope>, the policy will only succeed if the access token has scope "A". Otherwise, it will return an error.

```
<OAuthV2 async="false" continueOnError="false" enabled="true" name="OAuthV2-VerifyAccessTokenA">
    <DisplayName>Verify OAuth v2.0 Access Token</DisplayName>
    <ExternalAuthorization>false</ExternalAuthorization>
    <Operation>VerifyAccessToken</Operation>
    <Scope>A</Scope>
    <GenerateResponse enabled="true"/>
</OAuthV2>
```

To summarize, API developers are responsible for designing scope enforcement into their APIs. They do this by creating custom flows to handle specific scopes, and attaching VerifyAccessToken policies to enforce those scopes.

4.2.1.16.4 Code examples

Finally, let's take a look at some example API calls to help illustrate how tokens receive scopes and how scopes are enforced.

Default case

Let's say you have a developer app with products, and that the union of those products' scopes are: A, B, and C. This API call requests an access token, but does not specify a scope query parameter.

```
curl -X POST -H content-type:application/x-www-form-urlencoded http://wwitman-test.<host:port>/scopecheck1/token?grant_type=client_credentials
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

In this case, the generated token will be given scopes A, B, and C (the default behavior). The token's metadata would look something like this:

```
{
  "issued_at" : "1417016208588",
  "application_name" : "eb1a0333-5775-4116-9eb2-c36075ddc360",
  "scope" : "A B C",
  "status" : "approved",
  "api_product_list" : "[scopecheck1-yEgQbQqjRR]",
  "expires_in" : "1799",
  "developer.email" : "scopecheck1-yxiuHuZcDW@sap.com",
  "organization_id" : "0",
  "token_type" : "BearerToken",
  "client_id" : "atGFvl3jgA0pJd05rXKHeNAC69naDmpW",
  "access_token" : "MveXpj4UYXol38thNoJYIa8fBGlI",
  "organization_name" : "wwitman",
  "refresh_token_expires_in" : "0",
  "refresh_count" : "0"
}
```

Now, let's say you have an API endpoint that has scope "A" (that is, it's VerifyAccessToken requires scope "A"). Here's the VerifyAccessToken policy:

```
<OAuthV2 async="false" continueOnError="false" enabled="true" name="OAuthV2-
VerifyAccessTokenA">
    <DisplayName>Verify OAuth v2.0 Access Token</DisplayName>
    <ExternalAuthorization>false</ExternalAuthorization>
    <Operation>VerifyAccessToken</Operation>
    <Scope>A</Scope>
    <GenerateResponse enabled="true"/>
</OAuthV2>
```

Here's a sample call to and endpoint that enforces scope A:

```
curl -X GET -H Authorization: Bearer MveXpj4UYXol38thNoJYIa8fBGlI http://wwitman-
test.<host:port>/scopecheck1/resourceA
```

This GET call succeeds:

```
 {
   "hello" : "Tue, 25 Nov 2014 01:35:53 UTC"
 }
```

It succeeds because the VerifyAccessToken policy that is triggered when the endpoint is called requires scope A, and the access token was granted scopes A, B, and C -- the default behavior.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Filtering case

Let's say you have a developer app with products that have scopes A, B, C, and X. You request an access token and include the scope query parameter, like this:

```
curl -i -X POST -H content-type:application/x-www-form-urlencoded 'http://myorg-test.<host:port>/oauth/token?grant_type=client_credentials&scope=A X'
```

In this case, the generated token will be given scopes A and X, because both A and X are a valid scopes. Remember that the developer app recognizes scopes A, B, C, and X. In this case, you're filtering the list of API products based on these scopes. If a product has scope A or X, you can configure API endpoints that will enforce these scopes. If a product does not have scope A or X (let's say it has B,C, and Z), then the APIs that enforce scopes A or X cannot be called with the token.

When you call the API with the new token:

```
curl -X GET -H Authorization: Bearer RkmqoUkEIyIBCrtro1QpIG http://wwitman-test.<host:port>/scopecheck1/resourceX
```

The access token is validated by the API proxy. For example:

```
<OAuthV2 async="false" continueOnError="false" enabled="true" name="OAuthV2-VerifyAccessTokenX">
    <DisplayName>Verify OAuth v2.0 Access Token</DisplayName>
    <ExternalAuthorization>false</ExternalAuthorization>
    <Operation>VerifyAccessToken</Operation>
    <Scope>A X</Scope>
    <GenerateResponse enabled="true"/>
</OAuthV2>
```

The GET call triggers succeeds and it returns a response. For example:

```
 {
   "hello" : "Tue, 25 Nov 2014 01:35:53 UTC"
 }
```

It succeeds because the VerifyAccessToken policy requires scope A or X, and the access token includes scope A and X. Of course, if the <Scope>element were set to "B", this call would fail.

4.2.1.16.5 Summary

- It's important to understand how SAP API Management handles OAuth 2.0 scopes. Here are key takeaway points:
- A developer app "recognizes" the union of all scopes defined for all of its products.
- When an app requests an access token, it has the chance to specify which scopes it would like to have. It's up to SAP API Management (the authorization server) to figure out which scopes it will actually assign to the

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

access token based on (a) the scope(s) that are requested and (b) the ones that are recognized by the developer app.

- If SAP API Management is not configured to check for scope (the <Scope> element is missing from the VerifyAccessToken policy or it is empty), then the API call will succeed as long as the scope embedded in the access token matches one of the scopes recognized by the registered developer app (one of the scopes in the app's "master" list of scopes).
- If an access token does not have any scopes associated with it, then it will only succeed in cases where SAP API Management does not consider scope (the<Scope> element is missing from the VerifyAccessToken policy or it is empty).

4.2.1.17 Using third-party OAuth tokens

In this topic, we'll discuss how to use externally generated access and refresh tokens and authorization codes in SAP API Management.

4.2.1.17.1 About this topic

A common use case is where you have an existing OAuth system in place, and you would like to use the tokens generated by that system with SAP API Management. On the other hand, if you want to take advantage of SAP API management features like SAP API Management Analytics, the developer app ecosystem, developer portal, and so on, you need to use tokens that were generated with SAP API Management's OAuth system. This topic explains how to configure SAP API Management to work with third-party (non-SAP API Management) OAuth systems.

4.2.1.17.2 How to use third-party OAuth on SAP API Management

To use tokens from third-party OAuth systems in SAP API Management, you need to do these things:

- Configure the OAuthV2 policy that generates tokens with the <ExternalAuthorization> element set to true. If this element is false or not present, then SAP API Management validates the client_id and client_secret normally against the SAP API Management authorization store.
- Set the internal flow variable oauth_external_authorization_status to true. If this value is false (or if the variable is not present), SAP API management assumes that the third-party authorization failed and returns an error message.

Typically, this variable is set to true or false based on a service callout to a third-party authorization service. You can look at the service callout response and set the variable accordingly. Take a look at the ServiceCallout policy for details. Another technique for setting this variable is to use an AssignMessage policy with the AssignVariable element, like this:

```
<AssignMessage name="AssignMessage-SetVariable">
    <DisplayName>Assign Message - Set Variable</DisplayName>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <AssignVariable>
        <Name>oauth_external_authorization_status</Name>
        <Value>true</Value>
    </AssignVariable>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</AssignMessage>
```

- Configure one of these OAuthV2 elements: <ExternalAccessToken>, <ExternalRefreshToken>, or <ExternalAuthorizationCode>. These elements specify where SAP API Management should look to find the externally-generated access token or authorization code. Upon finding the external token or code, SAP API Management goes ahead and mints either an access token or an authorization code of its own. For example, this configuration tells SAP API Management to look for the token in a query parameter; however, it could be assigned to any flow variable accessible to the proxy.

  ```
  <ExternalAccessToken>request.queryparam.external_access_token</ExternalAccessToken>
  ```

  > **Note**
  >
  > Setting these elements is a common option when using <ExternalAuthorization>; however, you can also use these elements independently. For example, you can leave <ExternalAuthorization> as false (the default) so that the client_id and client_secret values are validated against SAP API Management, but still use an external access token from another system.

- Set the <StoreToken> element in OAuthV2 to true. If this is false, then the token will not be persisted.

For example, here we will mint an SAP API Management access token given that another valid token (from a third-party or legacy system) is provided in the query parameter `request.queryparam.external_access_token`.

```
<OAuthV2 name="OAuth-v20-Store-External-Token">
    <DisplayName>OAuth v2.0 1</DisplayName>
    <Attributes/>

<ExternalAccessToken>request.queryparam.external_access_token</ExternalAccessToken>
    <ExternalAuthorization>true</ExternalAuthorization>
    <Operation>GenerateAccessToken</Operation>
    <GenerateResponse enabled="true">
        <Format>FORM_PARAM</Format>
    </GenerateResponse>
    <ReuseRefreshToken>false</ReuseRefreshToken>
    <StoreToken>true</StoreToken>
    <SupportedGrantTypes>
        <GrantType>client_credentials</GrantType>
    </SupportedGrantTypes>
    <Tokens/>
</OAuthV2>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Then, for example, you can request an access token without specifying the client secret, because it is assumed that the client secret was already validated by the other system. Note that the client_id is still expected in the request and will be validated. By default, SAP API Management expects the client_id to be sent as x-www-form-urlencoded data. For example:

```
curl https://testmyapi-test.<host:port>/oauth-
delegated/generatetoken?external_access_token=123456 \
-d 'client_id=sxnS7SddD6494Akbqk74ej4SmvvqjL0O&grant_type=client_credentials'
```

In this case, SAP API Management will generate an access token, with an access token of 123456. The token metadata might look something like this. Notice the access_token property is 123456. And, as you can see, this token is wired up with SAP API Management-specific properties that will enable it to work with developers, developer apps, and products registered on SAP API Management.

```
{
  "issued_at" : "1415467907287",
  "application_name" : "9d3f4de-e501-4836-9bdd-f55e1d4b923",
  "scope" : "WRITE",
  "status" : "approved",
  "api_product_list" : "[PremiumWeatherAPI]",
  "expires_in" : "1799",
  "developer.email" : "tesla@weathersample.com",
  "organization_id" : "0",
  "token_type" : "BearerToken",
  "client_id" : "sxnS7Sdd6494Akb74ej4SmvvqjL0O",
  "access_token" : "123456"
  "organization_name" : "testmyapi",
  "refresh_token_expires_in" : "0",
  "refresh_count" : "0"
}
```

For a final example, you might call an API like this, with 123456 as the bearer token:

```
curl
'https://docs-test.<host:port>/oauth-
delegated/music?func=getSong&artist=radiohead&fmt=json'
-H 'Authorization:Bearer 123456'
```

In theory, you could apply this pattern with any third-party OAuth 2 authorization service, like Google or Twitter.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.18 Advanced OAuth 2.0 topics

4.2.1.18.1 Designating multiple callback URLs

When using the authorization code grant type, you must specify a callback URL when you register your developer app. The callback URL typically specifies the URL of an app that is designated to receive an authorization code on behalf of the client app. In addition, this URL string is used for validation. The client is required to send this URL to SAP API management when requesting authorization codes and access tokens, and the redirect_uri parameter must match the one that is registered. See also Requesting access tokens and authorization codes.

For example:

```
http://myorg-test.<host:port>/weather/oauth/authorizationcode?client_id=123456&response_type=code&redirect_uri=http://example.com/callbacak&scope=scope1%20scope2&state=abc
```

There is a use case for specifying multiple callback URLs in a single proxy application. For example, you may want to authenticate for multiple domains. For instance:

- http://myexample.com/callback
- http://myexample.uk/callback
- http://myexample.ja/callback

SAP API Management does not support specifying multiple callback URLs or using wildcard characters when a developer app is registered. To accomplish this use case, you can specify an empty callback URL when you register a developer app, and then put a logic in a JavaScript policy to validate incoming redirect URIs.

4.2.1.18.2 Auditing app end user consent

You may be required to verify that an app end user authorized an app. You can use the SAP API Management Audit API to do so.

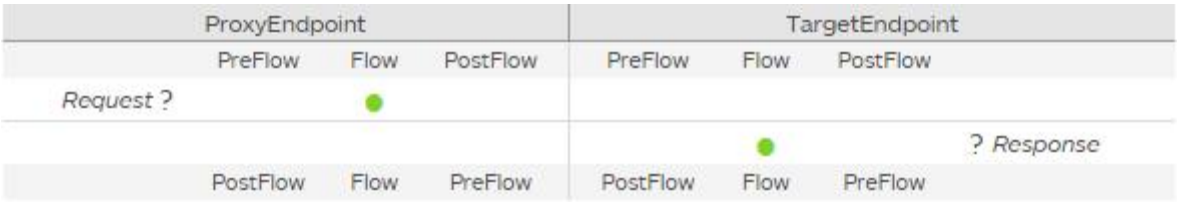
4.2.1.19 OAuthV2 policy

OAuthV2 is a multi-faceted policy for performing OAuth 2.0 grant type operations. This is the primary policy used to configure OAuth 2.0 endpoints on SAP API Management.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Where

Place this policy on custom API proxy flows to generate access tokens, refresh tokens, authorization codes, to perform token validation, and other operations. For example, you might put this policy on an API proxy endpoint called /authorizationcode for apps to request authorization codes. Likewise, you might put this policy on an API proxy endpoint called /token to generate an access token when that endpoint is called.



4.2.1.19.1 Samples

VerifyAccessToken

This OAuthV2 policy configuration (with the VerifyAccessToken operation) verifies that an access token submitted to SAP API Management is valid. When this policy operation is triggered, it looks for a valid access token in the request. If the access token is valid, the request is allowed to proceed. If invalid, all processing stops and an error is returned in the response.

```
<OAuthV2 async="false" continueOnError="false" enabled="true" name="OAuth-v20-2">
    <DisplayName>OAuth v2.0 2</DisplayName>
    <Operation>VerifyAccessToken</Operation>
    <AccessTokenPrefix>Bearer</AccessTokenPrefix> <!-- Optional, default is Bearer -->
</OAuthV2>
```

Note that only bearer (access) tokens are supported. MAC tokens are not supported.

Apps must pass the access token in the Authorization HTTP request header. For example:

```
$ curl -H "Authorization: Bearer ylSkZIjbdWybfsUQe9BqP0LH5Z" http://{org-name}-
{env}.<host:port>/weather/forecastrss?w=12797282
```

Element reference

The policy reference describes the elements and attributes of the OAuthV2 policy.

The way you configure this policy, and the elements you need to specify, depend on which operation you want the policy to perform. For example, if you are implementing the authorization code grant type, then you will require four separate OAuthV2 policies to perform authorization code generation, access code generation, access code validation, and refresh token generation. This reference lists all of the elements that can be used with this policy.

The sample policy shown below is one of many possible configurations. This sample shows an OAuthV2 policy configured for the GenerateAccessToken operation. It includes required and optional elements. Refer to the element descriptions in this section for details.

```
<OAuthV2 name="GenerateAccessToken">
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  <Operation>GenerateAccessToken</Operation>
  <ExpiresIn>1000</ExpiresIn>
  <GenerateResponse />
  <SupportedGrantTypes>
   <GrantType>authorization_code</GrantType>
  </SupportedGrantTypes>
  <GrantType>request.queryparam.grant_type</GrantType>
  <Code>request.queryparam.code</Code>
  <ClientId>request.queryparam.client_id</ClientId>
  <RedirectUri>request.queryparam.redirect_uri</RedirectUri>
  <Scope>request.queryparam.scope</Scope>
</OAuthV2>
```

Generating access tokens

For examples showing how to request access tokens for each of the supported grant types, see Requesting access tokens and authorization codes. The topic includes examples of these operations:

- Authorization code
- Client credentials
- Implicit
- Password
- Refresh Token
- Authorization code

Element reference

The policy reference describes the elements and attributes of the OAuthV2 policy.

The way you configure this policy, and the elements you need to specify, depend on which operation you want the policy to perform. For example, if you are implementing the authorization code grant type, then you will require four separate OAuthV2 policies to perform authorization code generation, access code generation, access code validation, and refresh token generation. This reference lists all of the elements that can be used with this policy.

The sample policy shown below is one of many possible configurations. This sample shows an OAuthV2 policy configured for the GenerateAccessToken operation. It includes required and optional elements. Refer to the element descriptions in this section for details.

```
<OAuthV2 name="GenerateAccessToken">
  <Operation>GenerateAccessToken</Operation>
  <ExpiresIn>1000</ExpiresIn>
  <GenerateResponse />
  <SupportedGrantTypes>
   <GrantType>authorization_code</GrantType>
  </SupportedGrantTypes>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  <GrantType>request.queryparam.grant_type</GrantType>
  <Code>request.queryparam.code</Code>
  <ClientId>request.queryparam.client_id</ClientId>
  <RedirectUri>request.queryparam.redirect_uri</RedirectUri>
  <Scope>request.queryparam.scope</Scope>
</OAuthV2>
```

Generate authorization code

For examples showing how to request authorization codes, see Requesting access tokens and authorization codes.

Element reference

The policy reference describes the elements and attributes of the OAuthV2 policy.

The way you configure this policy, and the elements you need to specify, depend on which operation you want the policy to perform. For example, if you are implementing the authorization code grant type, then you will require four separate OAuthV2 policies to perform authorization code generation, access code generation, access code validation, and refresh token generation. This reference lists all of the elements that can be used with this policy.

The sample policy shown below is one of many possible configurations. This sample shows an OAuthV2 policy configured for the GenerateAccessToken operation. It includes required and optional elements. Refer to the element descriptions in this section for details.

```
<OAuthV2 name="GenerateAccessToken">
  <Operation>GenerateAccessToken</Operation>
  <ExpiresIn>1000</ExpiresIn>
  <GenerateResponse />
  <SupportedGrantTypes>
   <GrantType>authorization_code</GrantType>
  </SupportedGrantTypes>
  <GrantType>request.queryparam.grant_type</GrantType>
  <Code>request.queryparam.code</Code>
  <ClientId>request.queryparam.client_id</ClientId>
  <RedirectUri>request.queryparam.redirect_uri</RedirectUri>
  <Scope>request.queryparam.scope</Scope>
</OAuthV2>
```

4.2.1.19.2 <OAuthV2> attributes

```
<OAuthV2 async="false" continueOnError="false" enabled="true" name="MyOAuthPolicy">
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The following attributes are common to all policy parent elements.

Attribute	Description	Default	Presence
name	The internal name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the <DisplayName> element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Required
continueOnError	Set to false to return an error when a policy fails. This is expected behavior for most policies. Set to true to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to true to enforce the policy. Set to false to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to true, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to true improves API proxy performance. However, overusing async can hurt performance with too much thread switching. To use asynchronous behavior in API proxies, see JavaScript callouts.	false	Optional

4.2.1.19.3 <DisplayName> element

Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

```
<DisplayName>Policy Display Name</DisplayName>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Default:	N/A If you omit this element, the the value of the policy'sname attribute is used.
Presence:	Optional
Type:	String

4.2.1.19.4 <AccessToken> element

```
<AccessToken>request.header.access_token</AccessToken>
```

By default, VerifyAccessToken expects the access token to be sent in an Authorization header. You can change that default using this element. For example request.queryparam.access_token indicates that the access token should be present as a query parameter.

If presented in the header, it must be an Authorization header and be sent as a Bearer token. For example:

-H "Authorization: Bearer Rft3dqrs56Blirls56a"

Currently, Bearer is the only supported prefix.

Default:	request.header.access_token
Presence:	Optional
Type:	String
Valid values:	A valid access token
Used with operations:	VerifyAccessToken

4.2.1.19.5 <AccessTokenPrefix> element

```
<AccessTokenPrefix>Bearer</AccessTokenPrefix>
```

By default, VerifyAccessToken expects the access token to be sent in an Authorization header as a Bearer token. For example:

```
-H "Authorization: Bearer Rft3dqrs56Blirls56a"
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Currently, Bearer is the only supported prefix.

Default:	Bearer
Presence:	Optional
Type:	String
Valid values:	Bearer
Used with operations:	VerifyAccessToken

4.2.1.19.6 <AppEndUser> element

```
<AppEndUser>request.queryparam.app_enduser</AppEndUser>
```

In cases where the app end user ID must be sent to the authorization server, this element lets you specify where SAP API Management should look for the end user ID. For example, it could be sent as a query parameter or in an HTTP header.

For example request.queryparam.app_enduser indicates that the AppEndUser should be present as a query parameter, as, for example, ?app_enduser=ntesla@theramin.com. To require the AppEndUser in an HTTP header, for example, set this value to request.header.app_enduser.

Providing this setting enables you to include an app end user ID in the access token. This feature is useful if you want to be able to retrieve or revoke OAuth 2.0 access tokens by end user ID.

Default:	N/A
Presence:	Optional
Type:	String
Valid values:	Any flow variable accessible to the policy at runtime.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Used with grant types:	· authorization_code · implicit · password · client_credentials

4.2.1.19.7 <Attributes/Attribute>

```
<Attributes>
    <Attribute name=”attr_name1” ref=”flow.variable”
display="true|false">value1</Attribute>
    <Attribute name=”attr_name2” ref=”flow.variable”
display="true|false">value2</Attribute>
</Attributes>
```

Use this element to add custom attributes to an access token or authorization code. For example, you may wish to embed a user ID or session identifier in an access token that can be extracted and checked at runtime.

This element takes a value from a flow variable or a default value which is specified in the policy. If both are specified, the value specified in the flow variable is taken. The optional display attribute can be set to true to false. If set to true, the attribute will not be shown in the response. Default value is false.

Default:	N/A
Presence:	Optional
Valid values:	name -The attribute name ref - The value of the attribute. Can come from a flow variable. display - (Optional) If set to true, the attribute will not be shown in the response. Default value is false.
Used with grant types:	· authorization_code · implicit · password · client_credentials · Can also be used with the GenerateAuthorizationCode operation.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.19.8 <ClientId> element

```
<ClientId>request.queryparam.client_id</ClientId>
```

In several cases, the client app must send the client ID to the authorization server. This element lets you specify where SAP API Management should look for the client ID. For example, it could be sent as a query parameter or in an HTTP header.

The variable request.queryparam.client_id indicates that the client_id should be present as a query parameter, as, for example, ?client_id=AfGlvs9. To require the ClientId in an HTTP header, for example, set this value to request.header.client_id.

Default:	request.formparam.client_id (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Any flow variable accessible to the policy at runtime
Used with grant types:	authorization_code password implicit client_credentials Can also be used with the GenerateAuthorizationCode operation.

4.2.1.19.9 <Code> element

```
<Code>request.queryparam.code</Code>
```

In the authorization grant type flow, the client must submit an authorization code to the authorization server (SAP API Management). This element lets you specify where SAP API Management should look for the authorization code. For example, it could be sent as a query parameter, HTTP header, or form parameter (the default).

The variable, request.queryparam.auth_code indicates that the authorization code should be present as a query parameter, as, for example, ?auth_code=AfGlvs9. To require the authorization code in an HTTP header, for example, set this value to request.header.auth_code.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Default:	request.formparam.code (a x-www-form-urlencoded and specified in the request body)
Presence:	optional
Type:	String
Valid values:	Any flow variable accessible to the policy at runtime
Used with grant types:	authorization_c

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.19.10 <ExpiresIn> element

```
<ExpiresIn>10000</ExpiresIn>
```

Enforces the expiry time of access tokens, refresh tokens, and authorization codes in milliseconds. The expiry time value is a system generated value plus the <ExpiresIn> value. If <ExpiresIn> is set to -1, the token or code is given an infinite lifetime. If <ExpiresIn> is not specified, the system applies a default value configured at the system level.

The expiry time can also be set at runtime using a reference to a flow variable. The flow variable can be retrieved from a header, query parameter, or form parameter (default). Or, it can be a hard-coded value.

For example request.queryparam.expires_in indicates that the expiry value should be present as a query parameter, as, for example, ?expires_in=360000. To require the value to come from an HTTP header, for example, set this value to request.header.expires_in.

The following stanza specifies a flow variable and a default value as well. Note that the flow variable value takes precedence over the specified default value.

```
<ExpiresIn ref="flow.variable">
    {default_value}
</ExpiresIn>
```

Default:	If not specified, the system applies a default value configured at the system level.
Presence:	Optional
Type:	Integer
Valid values:	Any integer, including -1 (which indicates an infinite expiry time).
Used with grant types:	· authorization_code · implicit · password · client_credentials · refresh_token Also used with GenerateAuthorizationCode operation.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.19.11 <ExternalAccessToken> element

```
<ExternalAccessToken>request.queryparam.external_access_token</ExternalAccessToken>
```

Tells SAP API Management where to find an external access token (an access token not generated by SAP API Management).

The variable request.queryparam.external_access_token indicates that the external access token should be present as a query parameter, as, for example, ?external_access_token=12345678. To require the external access token in an HTTP header, for example, set this value to request.header.external_access_token.

4.2.1.19.12 <ExternalAuthorization> element

```
<ExternalAuthorization>true</ExternalAuthorization>
```

If this element is false or not present, then SAP API Management validates the client_id and client_secret normally against the SAP API Management authorization store. Use this element when you want to work with third-party OAuth tokens.

Default:	false
Presence:	Optional
Type:	Boolean
Valid values:	true or false
Used with grant types:	· authorization_code · password · client_credentials

4.2.1.19.13 <GenerateResponse> element

```
<GenerateResponse enabled='true'/>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

If set to true, the policy generates and returns a response. If false, no response is sent. Instead, a set of flow variables are populated with values related to the policy's function. For example, a flow variable called oauthv2authcode.OAuthV2-GenerateAuthorizationCode.code gets populated with the newly minted auth code.

Default:	false
Presence:	Optional
Type:	string
Valid values:	true or false
Used with grant types:	· implicit · password · client_credentials · refresh_token Also can be used with the GenerateAuthorizationCode operation.

4.2.1.19.14 <GenerateErrorResponse> element

```
<GenerateErrorResponse enabled='true'/>
```

If set to true, the policy generates and returns a response if the ContinueOnError attribute is set to true.
If false (the default), no response is sent. Instead, a set of flow variables are populated with values related to the policy's function.

Default:	false
Presence:	Optional
Type:	string
Valid values:	true or false

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Used with grant types:	· implicit · password · client_credentials · refresh_token Also can be used with the GenerateAuthorizationCode operation.

4.2.1.19.15 <GrantType>

```
<GrantType>request.queryparam.grant_type</GrantType>
```

Tells the policy where to find the grant type parameter that is passed in a request. Per the OAuth 2.0 specification, the grant type must be supplied with requests for access tokens and authorization codes. The variable can be a header, query parameter, or form parameter (default). Or, it can be a hard-coded value.

For example request.queryparam.grant_type indicates that the Password should be present as a query parameter, as, for example, ?grant_type=password. To require the grant type in an HTTP header, for example, set this value to request.header.grant_type.

Default:	request.formparam.grant_type (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	string
Valid values:	A variable, as explained above, or a hard-coded grant type of: client_credentials, implicit, authorization_code, or password
Used with grant types:	· authorization_code · password · implicit · client_credentials · refresh_token

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.19.16 <Operation> element

<Operation>GenerateAuthorizationCode</Operation>

The OAuth 2.0 operation executed by the policy.

Default:	If <Operation> is not specified, SAP API Management looks at the list of <SupportedGrantTypes>. Only operations on those grant types will be successful. In other words, you can omit <Operation> if you specify a <GrantType> in the <SupportedGrantTypes> list. If neither <Operation> nor <SupportedGrantTypes> are specified, the default grant type is authorization_code. That is, authorization_code grant type requests will succeed, but all others will fail.
Presence:	Optional
Type:	String
Valid values:	· GenerateAccessToken - Generates an access token. · GenerateAccessTokenImplicitGrant - Generates an access token for the implict grant type. GenerateAuthorizationCode - Generates an auth code. Used with the authorization code grant type. RefreshAccessToken - Exchange a new access token for a refresh token. · VerifyAccessToken - Verifies that an access token sent in a request is valid. · InvalidateToken - Revokes an access token. After a token has been revoked, clients cannot use that token to call a protected API. · ValidateToken - Reinstates or "approves" an access token that was previously revoked.

4.2.1.19.17 <PassWord> element

```
<PassWord>request.queryparam.password</PassWord>
```

In cases where the app user's password must be sent to the authorization server, this element lets you specify where SAP API Management should look for the password. For example, it could be sent as a query parameter, in an HTTP header, or a form parameter (default).

For example request.queryparam.password indicates that the password should be present as a query parameter, as, for example, ?password=changeit. To require the password in an HTTP header, for example, set this value to request.header.password.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Default:	request.formparam.password (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Any flow variable available to the the policy at runtime.
Used with grant types:	password

4.2.1.19.18 <RedirectUri> element

<RedirectUri>request.queryparam.redirect_uri</RedirectUri>

Specifies where SAP API Management should look for the redirect_uri parameter in the request.

About redirection URIs

Redirection URIs are used with the authorization code and implicit grant types. The redirect URI tells the authorization server (SAP API Management) where to send an authorization code (for the auth code grant type) or access token (for the implicit grant type). It's important to understand when this parameter is required, when it is optional, and how it is used:

- (required) If a Callback URL is registered with the developer app that is associated with the request's client keys, and if the request_uri is present in the request, then the two must match exactly. If they do not match, an error is returned.
- (optional) If a Callback URL is registered, and the request_uri is missing from the request, then SAP API Management redirects to the registered Callback URL.
- (required) If a Callback URL is not registered, then the request_uri is required. Note that in this case, SAP API Management will accept ANY URL. This case can present a security issue, and therefore should only be used with trusted client apps. If client apps are not trusted, then the best practice is to always require the registration of a Callback URL.

You can send this parameter in a query parameter or in a header. The variable request.queryparam.redirect_uri indicates that the RedirectUri should be present as a query parameter, as, for example, ?redirect_uri=login.myapp.com. To require the RedirectUri in an HTTP header, for example, set this value to request.header.redirect_uri.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Default:	request.formparam.redirect_uri (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Any flow variable accessible in the policy at runtime
Used with grant types:	· authorization_code · implicit · Also used with the GenerateAuthorizationCode operation.

4.2.1.19.19 <RefreshToken> element

```
<RefreshToken>request.queryparam.refreshtoken</RefreshToken>
```

When requesting an access token using a refresh token, you must supply the refresh token in the request. This element lets you specify where SAP API Management should look for the refresh token. For example, it could be sent as a query parameter, HTTP header, or form parameter (the default).

The variable request.queryparam.refreshtoken indicates that the refresh token should be present as a query parameter, as, for example, ?refresh_token=login.myapp.com. To require the RedirectUri in an HTTP header, for example, set this value to request.header.refresh_token.

Default:	request.formparam.refresh_token (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Any flow variable accessible in the policy at runtime

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Used with grant types:	refresh_token

4.2.1.19.20 <RefreshTokenExpiresIn> element

```
<RefreshTokenExpiresIn>1000</RefreshTokenExpiresIn>
```

Enforces the expiry time of refresh tokens in milliseconds. The expiry time value is a system generated value plus the <RefreshTokenExpiresIn>value. If <RefreshTokenExpiresIn> is set to -1, the refresh token is given an infinite lifetime. If <RefreshTokenExpiresIn> is not specified, the system applies a default value configured at the system level.

The expiry time can also be set at runtime using a reference to a flow variable. The flow variable can be retrieved from a header, query parameter, or form parameter (default). Or, it can be a hard-coded value.

For example request.queryparam.expires_in indicates that the expiry value should be present as a query parameter, as, for example, ?expires_in=360000. To require the value to come from an HTTP header, for example, set this value to request.header.expires_in.

The following stanza specifies a flow variable and a default value as well. Note that the flow variable value takes precedence over the specified default value.

```
<RefreshTokenExpiresIn ref="flow.variable">
    {default_value}
</RefreshTokenExpiresIn>
```

Default:	If not specified, the system applies a default value configured at the system level.
Presence:	Optional
Type:	Integer
Valid values:	Any integer, including -1 (which indicates an infinite expiry time).

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Used with grant types:	· authorization_code · password · refresh_token

4.2.1.19.21 <ResponseType> element

<ResponseType>request.queryparam.response_type</ResponseType>

This element informs SAP API Management which grant type the client app is requesting. It is used only with the authorization code and implicit grant type flows.

By default, SAP API Management looks for the response type value in a response_type query parameter. If you wish to override this default behavior, use the <ResponseType> element to configure a flow variable containing the response type value. For example, if you set this element to request.header.response_type, SAP API Management looks for the response type to be passed in the request header.

Default:	request.formparam.response_type (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional. Use this element if you wish to override the default behavior.
Type:	String
Valid values:	Either code (for the authorization code grant type) or token (for the implicit grant type)
Used with grant types:	· implicit · Also used with the GenerateAuthorizationCode operation.

4.2.1.19.22 <ReuseRefreshToken> element

```
<ReuseRefreshToken>true</ReuseRefreshToken>
```

When set to true, the existing refresh token is reused until it expires. If false, a new refresh token is issued by SAP API Management when a valid refresh token is presented.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Default:	false
Presence:	optional
Type:	boolean
Valid values:	true or false
Used with grant type:	refresh_token

4.2.1.19.23 <Scope> element

```
<Scope>request.queryparam.scope</Scope>
```

If this element is present in an of the GenerateAccessToken or GenerateAuthorizationCode policies, it is used to specify which scopes to grant the token or code. These values are typically passed to the policy in the request from a client app. You can configure the element to take a flow variable, giving you the option to choose how the scopes are passed in a request. In the following example, request.queryparam.scope indicates that the scope should be present as a query parameter, as, for example, ?scope=READ. To require the scope in an HTTP header, for example, set this value to request.header.scope.

If this element appears in a "VerifyAccessToken" policy, then it is used to specify which scopes the policy should enforce. In this type of policy, the value must be a "hard coded" scope name -- you can't use variables. For example:

```
<Scope>A B</Scope>
```

Default:	No scope
Presence:	Optional
Type:	String

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Valid values:	If used with Generate* policies, a flow variable. If used with VerifyAccessToken, a space-separated list of scope names (strings).
Used with grant types:	· authorization_code · implicit · password · client_credentials · Can also be used with the GenerateAuthorizationCode and VerifyAccessToken operations.

4.2.1.19.24 <State> element

```
<State>request.queryparam.state</State>
```

In cases where the client app must send the state information to the authorization server, this element lets you specify where SAP API Mangement should look for the state values. For example, it could be sent as a query parameter or in an HTTP header. The state value is typically used as a security measure to prevent CSRF attacks.

For example request.queryparam.state indicates that the state should be present as a query parameter, as, for example, ?state=HjoiuKJH32. To require the state in an HTTP header, for example, set this value to request.header.state.

Default:	No state
Presence:	Optional
Type:	String
Valid values:	Any flow variable accessible to the policy at runtime
Used with grant types:	· All · Can also be used with the GenerateAuthorizationCode operation

4.2.1.19.25 <StoreToken> element

```
<StoreToken>true</StoreToken>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Set this element to true when the <ExternalAuthorization> element is true. The <StoreToken> element tells SAP API Management to store the external access token. Otherwise, it will not be persisted.

Default:	false
Presence:	Optional
Type:	Boolean
Valid values:	true or false
Used with grant types:	· authorization_code · password · client_credentials

4.2.1.19.26 <SupportedGrantTypes> / <GrantType> element

```
<SupportedGrantTypes>
    <GrantType>authorization_code</GrantType>
    <GrantType>client_credentials</GrantType>
    <GrantType>implicit</GrantType>
    <GrantType>password</GrantType>
</SupportedGrantTypes>
```

Specifies the grant types supported by an OAuth token endpoint on SAP API Management. An endpoint may support multiple grant types (that is, a single endpoint can be configured to distribute access tokens for multiple grant types.) For more on endpoints, see Understanding OAuth endpoints. The grant type is passed in token requests in a grant_type parameter.

If no supported grant types are specified, then the only allowed grant types are authorization_code and implicit. See also the <GrantType>element (which is a higher-level element that is used to specify where SAP API Management should look for the grant_type parameter that is passed in a client request. SAP API Management will make sure that the value of the grant_type parameter matches one of the supported grant types).

Default:	authorization _code and implicit

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Presence:	Required
Type:	String
Valid values:	· client_credentials · authorization_code · password · implicit

4.2.1.19.27 <Tokens>/<Token> element

Used with the ValidateToken and InvalidateToken operations.

The <Token> element identifies the flow variable that defines the source of the token to be revoked. If developers are expected to submit access tokens as query parameters named access_token, for example, use request.queryparam.access_token.

4.2.1.19.28 <UserName> element

```
<UserName>request.queryparam.user_name</UserName>
```

In cases where the app user name must be sent to the authorization server, this element lets you specify where SAP API Management should look for the end user name. For example, it could be sent as a query parameter or in an HTTP header.

For example request.queryparam.username indicates that the username should be present as a query parameter, as, for example, ?username=joe. To require the UserName in an HTTP header, for example, set this value to request.header.username.

Default:	request.formparam.password (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Valid values:	Any variable setting.
Used with grant types:	password

4.2.1.19.29 Verifying access tokens

Once a token endpoint is set up for an API proxy, a corresponding OAuthV2 policy that specifies the VerifyAccessToken operation is attached to the Flow that exposes the protected resource.

For example, to ensure that all requests to an API are authorized, the following policy enforces access token verification:

```
<OAuthV2 name="VerifyOAuthAccessToken">
  <Operation>VerifyAccessToken</Operation>
</OAuthV2>
```

The policy is attached to the API resource to be protected. To ensure that all requests to an API are verified, attach the policy to the ProxyEndpoint request PreFlow, as follows:

```
<PreFlow>
  <Request>
    <Step><Name>VerifyOAuthAccessToken</Name></Step>
  </Request>
</PreFlow>
```

The following optional elements can be used to override the default settings for the VerifyAccessToken operation.

Name	Description
Scope	A space-delimited list of scopes. Verification will succeed if at least one of the scopes listed is present in the access token. For example, the following policy will check the access token to ensure that it contains at least one of the scopes listed. If READ or WRITE is present, verification will succeed. <OAuthV2 name="ValidateOauthScopePolicy"> <Operation>VerifyAccessToken</Operation> <Scope>READ WRITE</Scope> </OAuthV2>

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description
AccessToken	The variable where the access token is expected to be located. For examplerequest.queryparam.accesstoken. By default, the access token is expected to be presented by the app in the Authorization HTTP header, according to the OAuth 2.0 specification. Use this setting if the access token is expected to be presented in a non-standard location, such as a query parameter, or an HTTP header with a name other than Authorization.

4.2.1.19.30 Specifying request variable locations

For each grant type, the policy makes assumptions about the location or required information in request messages. These assumptions are based on the the OAuth 2.0 specification. If your apps need to deviate from the OAuth 2.0 specification, then you can specify the expected locations for each parameter. For example, when handling an authorization code, you can specify the location of the authorization code, the client ID, the redirect URI, and the scope. These can be specified as HTTP headers, query parameters, or form parameters.

The example below demonstrates how you can specify the location of required authorization code parameters as HTTP headers:

```
...
  <GrantType>request.header.grant_type</GrantType>
  <Code>request.header.code</Code>
  <ClientId>request.header.client_id</ClientId>
  <RedirectUri>request.header.redirect_uri</RedirectUri>
  <Scope>request.header.scope</Scope>
  ...
```

Or, if necessary to support your client app base, you can mix and match headers and query parameters:

```
  ...
  <GrantType>request.header.grant_type</GrantType>
  <Code>request.header.code</Code>
  <ClientId>request.queryparam.client_id</ClientId>
  <RedirectUri>request.queryparam.redirect_uri</RedirectUri>
  <Scope>request.queryparam.scope</Scope>
  ...
```

Only one location can be configured per parameter.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.19.31 Flow variables

The flow variables defined in this table are populated when the respective OAuth policies are executed, and hence are available to other policies or applications executing in the API proxy flow.

4.2.1.19.32 VerifyAccessToken operation

These variables are set when the VerifyAccessToken policy operation executes.

API product variables will be populated automatically if the API products are configured with valid environment, proxies, and API resources (derived from the proxy.pathsuffix). Explicitly setting flow.resource.name variable is not required.

Where the API products are not configured with valid environments and API proxies, then flow.resource.name must explicitly be set to populate API product variables in the flow.

Variables	Description
organization_name	The name of the organization where the proxy is executing.
developer.id	The ID of the developer associated with the registered client app.
developer.app.name	The name of the developer associated with the registered client app.
client_id	The client ID of the registered client app.
grant_type	The grant type associated with the request.
token_type	The token type associated with the request.
access_token	The access token that is being verified.
accesstoken.{custom_attribute}	A named custom attribute in the access token.
issued_at	The date the access token was issued.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

expires_in	The expiration time for the access token.
status	The status of the access token (e.g., approved or revoked).
scope	The scope (if any) associated with the access token.
apiproduct.<custom_attribute_name>	A named custom attribute of the API product associated with the registered client app.
apiproduct.name	The name of the API product associated with the registered client app.
App-specific variables	
app.name	
app.id	
app.accessType	
app.callbackUrl	
app.status	
app.scopes	
app.appFamily	
app.apiproducts	
app.appParentStatus	
app.appType	
app.appParentId	

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

app.created_by	
app.created_at	
app.last_modified_at	
app.last_modified_by	
app.{custom_attributes}	
Developer-specific variables	Note : If the app.appType is "Company" ,then company attributes are populated and if app.appType is "Developer", then developer attributes are populated.
developer.id	
developer.userName	
developer.firstName	
developer.lastName	
developer.email	
developer.status	
developer.apps	
developer.created_by	
developer.created_at	
developer.last_modified_at	
developer.last_modified_by	

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

developer.{custom_attributes}	
Company-specific variables	Note : If the app.appType is "Company" ,then company attributes are populated and if app.appType is "Developer", then developer attributes are populated.
company.id	
company.displayName	
company.apps	
company.appOwnerStatus	
company.created_by	
company.created_at	
company.last_modified_at	
company.last_modified_by	
company.{custom_attributes}	

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.19.33 GenerateAuthorizationCode operation

These variables are set when the GenerateAuthorizationCode policy operation executes successfully:

Variable	Description
oauthv2authcode.{policy_name}.code	The authorization code generated when the policy executes.
oauthv2authcode.{policy_name}.redirect_uri	The redirect URI associated with the registered client app.
oauthv2authcode.{policy_name}.scope	The optional OAuth scope passed in the client request.
oauthv2authcode.{policy_name}.client_id	The client ID passed in the client request.

4.2.1.19.34 GenerateAccessToken

These variables are set when the GenerateAccessToken policy operation executes successfully for the authorization code, password, and client credentials grant type flows:

Variable	Description
oauthv2accesstoken.{policy_name}.access_token	The access token generated when the policy executes.
oauthv2accesstoken.{policy_name}.token_type	Will be set to accesstoken.
oauthv2accesstoken.{policy_name}.expires_in	The expiry value for the token. See the <ExpiresIn> element for details.
oauthv2accesstoken.{policy_name}.refresh_token	The refresh token generated when the policy executes.
oauthv2accesstoken.{policy_name}.refresh_token_expires_in	The lifespan of the refresh token, in seconds.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Variable	Description
oauthv2accesstoken.{policy_name}.refresh_token_issued_at	This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.
oauthv2accesstoken.{policy_name}.refresh_token_status	Set to approved or revoked.

4.2.1.19.35 GenerateAccessTokenImplicitGrant

These variables are set when the GenerateAccessTokenImplicit policy operation executes successfully for the implicit grant type flow:

Variable	Description
oauthv2accesstoken.{policy_name}.access_token	The access token generated when the policy executes.
oauthv2accesstoken.{policy_name}.expires_in	The expiry value for the token. See the <ExpiresIn> element for details.

4.2.1.19.36 RefreshAccessToken operation

These variables are set when the RefreshAccessToken policy operation executes successfully:

Variable	Description
oauthv2accesstoken.{policy_name}.access_token	The access token generated when the policy executes.
oauthv2accesstoken.{policy_name}.token_type	Will be set to accesstoken.
oauthv2accesstoken.{policy_name}.expires_in	The expiry value for the token. See the <ExpiresIn> element for details.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Variable	Description
oauthv2accesstoken.{policy_name}.refresh_token	The refresh token generated when the policy executes.
oauthv2accesstoken.{policy_name}.refresh_token_expires_in	The lifespan of the refresh token, in seconds.
oauthv2accesstoken.{policy_name}.refresh_token_issued_at	This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.
oauthv2accesstoken.{policy_name}.refresh_token_status	Set to approved or revoked.

4.2.1.19.37 GenerateErrorResponse

These variables are set when the `GenerateErrorResponse` element is true.

Variable	Description
oauthV2.{policy-name}.failed	Set to true if the policy failed.
oauthv2.{policy_name}.{fault_name}	The name of the fault. For example, invalid_request
oauthv2.{policy_name}.{fault_cause}	The fault reason. For example: Token Expired

4.2.1.19.38 Error codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  "contexts" : [ ]
}
```

The OAuthV2 Policy type defines the following error codes.

For OAuth-related HTTP error codes, see OAuth HTTP status code reference.

Error Code	Message
FailedToResolveOAuthConfig	Failed to resolve oauth config {0}
OperationRequired	Operation required
InvalidOperation	Operation {0} is invalid
InsufficientScope	Required scope(s) : {0}
FailedToResolveClientId	Failed to resolve client id variable {0}
FailedToResolveAccessToken	Failed to resolve access token variable {0}
FailedToResolveRefreshToken	Failed to resolve refresh token variable {0}
FailedToResolveToken	Failed to resolve token variable {0}
FailedToResolveAuthorizationCode	Failed to resolve authorization code variable {0}
ExpiresInNotApplicableForOperation	ExpiresIn element is not valid for operation {0}
RefreshTokenExpiresInNotApplicableForOperation	RefreshToken ExpiresIn element is not valid for operation {0}
InvalidValueForExpiresIn	Invalid value for ExpiresIn element for operation {0}
InvalidValueForRefreshTokenExpiresIn	Invalid value for Refresh Token ExpiresIn element for operation {0}
InvalidGrantType	Invalid grant type: {0}

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Error Code	Message
GrantTypesNotApplicableForOperation	Grant types are not applicable for operation {0}
UnSupportedGrantType	Unsupported Grant Type: {0}
InvalidParameter	Invalid required paramater: {0}
MissingParameter	Missing required paramater: {0}
SpecifyValueOrRefApiKey	Specify Api Key as value or ref in stepDefinition {0}
SpecifyAPIProduct	Specify the API Product to get details {0}
FailedToFetchApiProduct	Failed to fetch api product for key {0}
FailedToResolveAPIKey	Failed to resolve API Key variable {0}
FailedToResolveAPIProduct	Failed to resolve API product variable {0}
InvalidTokenType	Valid token types : {0}, Invalid token type {1} in stepDefinition {2}
TokenValueRequired	Token value is required in stepDefinition {0}
FailedToResolveRealm	Failed to resolve realm {0}

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

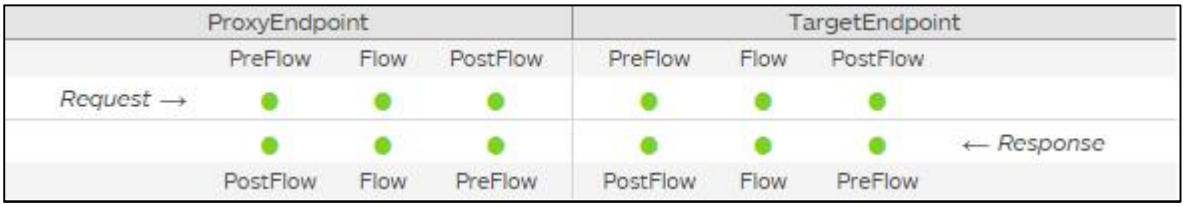
4.2.1.20 Get OAuth V2 Info policy

What

Gets attributes of access tokens, refresh tokens, and authorization codes and populates variables with the values of those attributes. This policy type can be useful when you need to configure dynamic, conditional behavior based on a value in a token or code. Whenever token validation occurs, variables are automatically populated with the values of token attributes. However, in cases where token validation has not occured, you can use this feature to explicity populate variables with the attribute values of a token.

Where

This policy can be attached in the following locations. See the Samples section below for specific attachment guidance in different situations.



4.2.1.20.1 Samples

Access token

This example shows how to retrieve profile information for an access token and use it to populate a set of variables. In this case, the policy expects to find the access token in a query parameter called access_token. Given the access token, the policy looks up the token's profile and populates a set of variables with the profile data. The variables will be prefixed with oauthv2accesstoken.

```
<GetOAuthV2Info name="GetTokenAttributes">
  <AccessToken ref="request.queryparam.access_token"></AccessToken>
</GetOAuthV2Info>
```

You can then access the variables using JavaScript or another means. For example, to retrieve the scope(s) associated with an access token using JavaScript:

```
var scope = context.getVariable('oauthv2accesstoken.GetTokenAttributes.scope');
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Auth code

Retrieving authorization code attributes

As with access tokens (described in the Access Code example), you can retrieve authorization code attributes by using the AuthorizationCodeelement in the policy as follows:

```
<GetOAuthV2Info name="GetAuthCodeAttributes">
    <AuthorizationCode ref="{variable name}"/>
</GetOAuthV2Info>
```

For example:

```
<GetOAuthV2Info name="GetAuthCodeAttributes">
    <AuthorizationCode ref="request.queryparam.code"></AuthorizationCode>
</GetOAuthV2Info>
```

Refresh token

Retrieving refresh token attributes

As with access tokens (described in the Access Code example), you can retrieve refresh token attributes by using the AuthorizationCode element in the policy as follows:

```
<GetOAuthV2Info name="GetTokenAttributes">
    <RefreshToken ref="{variable name}"/>
</GetOAuthV2Info>
```

For example:

```
<GetOAuthV2Info name="GetTokenAttributes">
  <RefreshToken ref="request.queryparam.refresh_token"></RefreshToken>
</GetOAuthV2Info>
```

Static

In some rare cases you may need to get the profile of a statically configured token (one that is not accessible through a variable). You can do this by providing the value of the access token as an element.

```
<GetOAuthV2Info name="GetTokenAttributes">
  <AccessToken>shTUmeI1geSKin0TODcGLXBNe9vp</AccessToken>
</GetOAuthV2Info>
```

You can do this with all other token types (client ID, authorization code, and refresh tokens) as well.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.20.2 Element Reference

The element reference describes the elements and attributes of the GetOAuthV2Info policy.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetOAuthV2Info async="false" continueOnError="false" enabled="true"
name="GetOAuthV2Info-1"
    <DisplayName>Get OAuth v2.0 Info 1</DisplayName>
    <AccessToken ref={some-variable}></AccessToken>
</GetOAuthV2Info
```

4.2.1.20.3 <GetOAuthV2Info> attributes

```
<GetOAuthV2Info async="false" continueOnError="false" enabled="true" name="Get-OAuth-
v20-Info-1">
```

The following attributes are common to all policy parent elements.

Attribute	Description	Default	Presence
Name	The internal name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the <DisplayName> element to label the policy in the management UI proxy editor with a different, natural-language name.	NA	Required
continueOnError	Set to `false` to return an error when a policy fails. This is expected behavior for most policies. Set to `true` to have flow execution continue even after a policy fails.	False	Optional
Enabled	Set to `true` to enforce the policy. Set to `false` to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	True	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to `true`, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to true improves API proxy	False	Optional

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Attribute	Description	Default	Presence
	performance. However, overusing async can hurt performance with too much thread switching. To use asynchronous behavior in API proxies, see JavaScript callouts.		

4.2.1.20.4 <DisplayName> element

Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

```
<DisplayName>Policy Display Name</DisplayName>
```

Default: N/A
If you omit this element, the the value of the policy'sname attribute is used.

Presence: Optional

Type: String

4.2.1.20.5 <AccessToken> element

Retrieves the profile for an access token. You pass in a either a variable that contains the access token string or a literal token string (rare case). In this example, the access token is retrieved from a query parameter passed in a request.

```
<AccessToken ref="request.queryparam.access_token"></AccessToken>
```

Default:	request.formparam.access_token (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Either a flow variable containing an access token string, or a literal string.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.20.6 <AuthorizationCode> element

Retrieves the profile for an authorization code. You pass in a either a variable that contains the auth code string or a literal token string (rare case). In this example, the auth code is retrieved from a query parameter passed in a request. For a list of variables populated by this operation, see "Flow variables".

```
<AuthorizationCode ref="request.queryparam.authorization_code"></AuthorizationCode>
```

Default:	request.formparam.access_token (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Either a flow variable containing an auth code string, or a literal string.

4.2.1.20.7 <ClientId> element

Retrieves information related to a client ID. In this example, the client ID is retrieved from a query parameter passed in a request. For a list of variables populated by this operation, see "Flow variables".

```
<ClientId>ref="request.queryparam.client_id"></ClientId>
```

Default:	request.formparam.access_token (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Either a flow variable containing an auth code string, or a literal string.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.20.8 <RefreshToken> element

Retrieves the profile for a refresh token. You pass in a either a variable that contains the refresh token string or a literal token string (rare case). In this example, the refresh token is retrieved from a query parameter passed in a request. For a list of variables populated by this operation, see "Flow variables".

```
<RefreshToken ref="request.queryparam.refresh_token"></RefreshToken>
```

Default:	request.formparam.access_token (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Either a flow variable containing an refresh token string, or a literal string.

4.2.1.20.9 Flow variables

The GetOAuthV2Info policy populates these variables, and is typically used in cases where you need the profile data, but where a grant or validation has not occurred yet. .

4.2.1.20.10 Client ID variables

These variables are populated when the <ClientId> operation executes:

oauthv2client.{policy_name}.client_id

oauthv2client.{policy_name}.client_secret

oauthv2client.{policy_name}.redirection_uris (* Note the spelling here -- 'redirection_uris')

oauthv2client.{policy_name}.developer.email

oauthv2client.{policy_name}.developer.app.name

oauthv2client.{policy_name}.developer.id

oauthv2client.{policy_name}.{custom_attribute_name}

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.20.11 Access token variables

These variables are populated when the <AccessToken> operation executes:

oauthv2accesstoken.{policy_name}.access_token

oauthv2accesstoken.{policy_name}.scope

oauthv2accesstoken.{policy_name}.refresh_token

oauthv2accesstoken.{policy_name}.accesstoken.{custom_attribute_name}

oauthv2accesstoken.{policy_name}.developer.id

oauthv2accesstoken.{policy_name}.developer.app.name

oauthv2accesstoken.{policy_name}.expires_in

oauthv2accesstoken.{policy_name}.status

4.2.1.20.12 Authorization code variables

These variables are populated when the <AuthorizationCode> operation executes:

oauthv2authcode.{policy_name}.client_id

oauthv2authcode.{policy_name}.organization_id

oauthv2authcode.{policy_name}.issued_at

oauthv2authcode.{policy_name}.expires_in

oauthv2authcode.{policy_name}.redirect_uri

oauthv2authcode.{policy_name}.status

oauthv2authcode.{policy_name}.state

oauthv2authcode.{policy_name}.scope

oauthv2authcode.{policy_name}.id

oauthv2authcode.{policy_name}.{custom_attribute_name}

4.2.1.20.13 Refresh token variables

These variables are populated when the <RefreshToken> operation executes:

oauthv2authcode.{policy_name}.access_token

oauthv2authcode.{policy_name}.refresh_token

oauthv2authcode.{policy_name}.client_id

oauthv2authcode.{policy_name}.refresh_count

oauthv2authcode.{policy_name}.organization_name

oauthv2authcode.{policy_name}.refresh_token_expires_in

oauthv2authcode.{policy_name}.refresh_token_issued_at

oauthv2authcode.{policy_name}.refresh_token_status

oauthv2authcode.{policy_name}.developer.email

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
oauthv2authcode.{policy_name}.developer.id
oauthv2authcode.{policy_name}.developer.app.name
oauthv2authcode.{policy_name}.developer.app.id
oauthv2authcode.{policy_name}.{custom_attribute_name}
```

4.2.1.20.14 Schema

Each policy type is defined by an XML schema (.xsd).

4.2.1.21 Set OAuth V2 Info policy

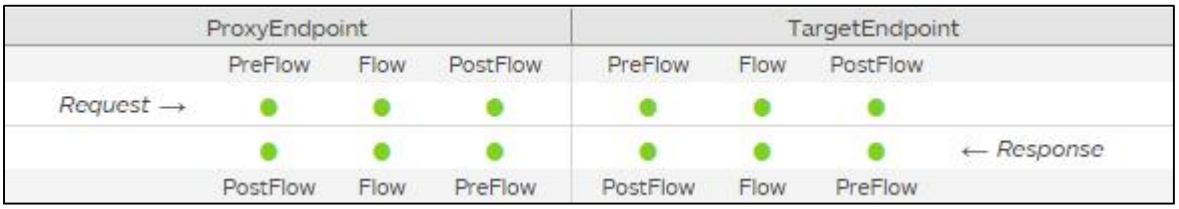
What

Lets you add or update custom attributes associated with an access token, refresh token, or authorization code. Custom attributes might include things like department name, a customer ID, or a session identifier.

You can only add or modify custom attributes. You cannot use this policy to change fields like scope, status, expires_in, developer_email, client_id, org_name, or refresh_count. If an attribute already exists, this policy updates it. If it does not exist, the policy adds it. The access token referenced must be valid and in an approved state.

Where

This policy can be attached in the following locations.



4.2.1.21.1 Samples

Below is an example policy used to update an OAuth 2.0 access token. The example below locates the access token on the request message by looking for a query parameter called access_token. When an access token is presented by a client app, the policy below will locate the access token in the query parameter. It will then update the access token's profile in two ways: it will added a custom property called department.id to the profile. It will also modify the access token's scope property to the value READ, WRITE.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<SetOAuthV2Info name="SetOAuthV2Info">
  <AccessToken ref="request.queryparam.access_token"></AccessToken>
  <Attributes>
    <Attribute name="department.id"
ref="request.queryparam.department_id"></Attribute>
    <Attribute name="scope" ref="">READ, WRITE</Attribute>
  </Attributes>
</SetOAuthV2Info>
```

> **Note**
>
> If an attribute already exists in the access token profile, then it will be updated with the new value in the policy. If an attribute does not exist, then the attribute will be added to the access token's profile.

4.2.1.21.2 Element Reference

The element reference describes the elements and attributes of the SetOAuthV2 policy.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SetOAuthV2Info async="false" continueOnError="false" enabled="true"
name="SetOAuthV2Info-1">
    <DisplayName>Set OAuth v2.0 Info 1</DisplayName>
    <AccessToken ref={some-variable}></AccessToken>
    <Attributes/>
</SetOAuthV2Info
```

4.2.1.21.3 <SetOAuthV2Info> attributes

```
<SetOAuthV2Info async="false" continueOnError="false" enabled="true" name="Set-OAuth-
v20-Info-1">
```

The following attributes are common to all policy parent elements.

Attribute	Description	Default	Presence
name	The internal name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.	N/A	Required

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

	Optionally, use the <DisplayName> element to label the policy in the management UI proxy editor with a different, natural-language name.		
continueOnError	Set to false to return an error when a policy fails. This is expected behavior for most policies. Set to true to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to true to enforce the policy. Set to false to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to true, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to true improves API proxy performance. However, overusing async can hurt performance with too much thread switching. To use asynchronous behavior in API proxies, see JavaScript callouts.	false	Optional

4.2.1.21.4 DisplayName> element

Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

```
<DisplayName>Policy Display Name</DisplayName>
```

Default: N/A

If you omit this element, the the value of the policy'sname attribute is used.

Presence: Optional

Type: String

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.21.5 <AccessToken> element

Identifies the variable where the access token is located. For example, if the access token is attached to request message as a query parameter, specify request.queryparam.access_token. You can use any valid variable that references the token. Or, could pass in the literal token string (rare case).

```
<AccessToken ref="request.queryparam.access_token"></AccessToken>
```

Default: N/A

Presence: Required

Type: String

4.2.1.21.6 Attributes

Attribute	Description
ref	An access token variable. Typically, retrieved from a flow variable.

4.2.1.21.7 <Attributes> element

A set of attributes in the access token profile that will be modified or augmented.

Default: N/A

Presence: Required

Type: N/A

4.2.1.21.8 <Attributes> / <Attribute> element

An individual attribute to update.

The name attribute identifies the property of the access token profile to be updated. For example, to modify the access token's scope property, specify scope as the value of the name attribute.

The ref attribute specifies either variable or a static whose value will be used as the value of the access token profile property that will be updated. For example to update the attribute scope with the value READ, WRITE:

```
<Attributes>
    <Attribute name="department.id"
ref="request.queryparam.department_id"></Attribute>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  <Attribute name="scope" ref="">READ, WRITE</Attribute>
</Attributes>
```

Default: N/A

Presence: Optional

Type: N/A

4.2.1.21.9 Attributes

Attribute	Description	Default	Presence
name	The name of the profile attribute to add or change.	N/A	
ref	The value to assign to the profile attribute.	N/A	Optional

4.2.1.21.10 Flow variables

On success, the following flow variables will be set:

```
oauthv2accesstoken.{policyName}.access_token
oauthv2accesstoken.{policyName}.client_id
oauthv2accesstoken.{policyName}.refresh_count
oauthv2accesstoken.{policyName}.organization_name
oauthv2accesstoken.{policyName}.expires_in
oauthv2accesstoken.{policyName}.refresh_token_expires_in
oauthv2accesstoken.{policyName}.issued_at
oauthv2accesstoken.{policyName}.status
oauthv2accesstoken.{policyName}.api_product_list
oauthv2accesstoken.{policyName}.token_type
oauthv2accesstoken.{policyName}.{custom_attribute_name}
On failure, following variable will be set:
oauthv2accesstoken.{policyName}.failed: true
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.1.21.11 Schema

Each policy type is defined by an XML schema (.xsd).

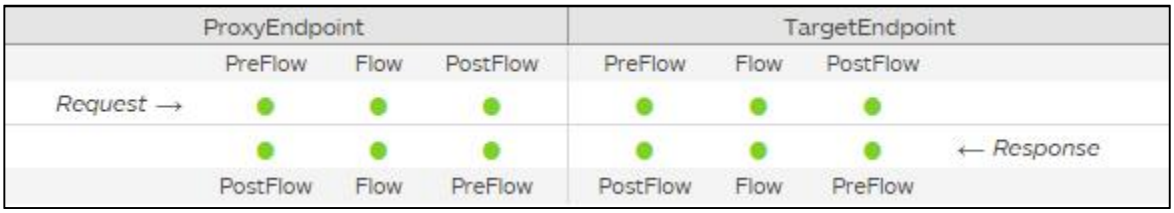
4.2.2 OAuth v1.0a policy

What

OAuth v1.01 defines a standard protocol that enables app users to authorize apps to consume APIs on their behalf, without requiring app users to disclose their passwords to the app in the process. SAP API Management enables you to protect APIs in a way that ensures that an app uses has authorized the app to consume an API. SAP API Management also provides policy-based functionality for configuring the endpoints that app developers can use to obtain access tokens.

Where

This policy can be attached in the following locations, but see the notes following the table for specific guidance.



4.2.2.1 Usage notes

SAP API Management provides an OAuth v1.0a policy type that enables you to authorize apps with OAuth 1.0a. The OAuthV1 policy type is responsible for generating request tokens, generating access tokens, and verifying access tokens based on the OAuth 1.0a specification.

Once you have configured OAuth policies on your API, apps must obtain access tokens to consume them. To do so, the app must exchange a request token for an access token. Rather than relying on a single password as the master key for every app that accesses an API, OAuth uses this token to provide “delegated authentication” between APIs and apps. The resource owner can issue access tokens with restricted scope and limited lifetime, and revoke them independently.

On token requests, SAP API Management OAuth 1.0a only supports HMAC-SHA signature algorithm for the oauth_signature_method header parameter, as described in the OAuth 1.0a specification.

The name attribute for this policy is restricted to these characters: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.2.2.2 Generating a request token

A request token is used by a consumer app to obtain authorization from the app end user, and is then presented to the 'token endpoint' to be exchanged for an access token. A request token is generated from a valid consumer key. Here are examples of the simple and comprehensive forms for generating request tokens.

Simple form

```
<OAuthV1 name="OAuthV1-GenerateRequestToken-1">
  <Operation>GenerateRequestToken</Operation>
</OAuthV1>
```

Comprehensive form

```
<OAuthV1 name="OAuthV1-GenerateRequestToken-1">
  <Operation>GenerateRequestToken</Operation>
    <URL ref="flow.variable">value</URL>
    <GenerateResponse enabled="true">
      <Format>FORM_PARAM | XML</Format>
    </GenerateResponse>
    <GenerateErrorResponse enabled="true">
      <Format>FORM_PARAM | XML</Format>
      <Realm>http://oauth.sap.com/oauth/1/</Realm>
    </GenerateErrorResponse>
</OAuthV1>
```

The policy enforces the following OAuth semantics:

- The consumer key is valid.
- The signature is valid.

A successful request returns the following responses:

- Generates a request token and its attributes in the flow oauth_token, oauth_token_secret, oauth_callback_confirmed, oauth_response
- Also makes a consumer token and its attributes available in the flow oauth_consumer_key, oauth_consumer_secret

The <URL> element allows proper OAuthV1 signature calculation when running behind Elastic Load Balancers (ELBs). The ref attribute takes precedence when both the ref attribute and a text value are specified. If ref is not provided, or if the flow variable given in ref is NULL, then the text value will be considered for generating the signature. For example:

```
<URL ref="secretURL">https://example-test.<host:port>/oauth/access_token</URL>
```

On failure, a request returns the appropriate response status code with an error message.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Sample request:

```
https://$org-$env.$api_domain/oauth1-proxy/request_token -H 'Authorization: OAuth
oauth_callback="oob", oauth_signature_method="HMAC-SHA1", oauth_token="\",
oauth_consumer_key="{consumerkey}", oauth_timestamp="{timestamp}",
oauth_nonce="{nonce}", oauth_version="1.0", oauth_signature="{signature}"'
```

Generating an access token

An access token is used by the consumer to gain access to the protected resources on behalf of the user, instead of using the user's service provider credentials. An access token is created with a valid key, request token, and verifier combination. Here are examples of the simple and comprehensive forms for generating access tokens.

Simple form

```
<OAuthV1 name="OAuthV1-GenerateAccessToken-1">
   <Operation>GenerateAccessToken</Operation>
</OAuthV1>
```

Comprehensive form

```
<OAuthV1 name="OAuthV1-GenerateAccessToken-1">
  <Operation>GenerateAccessToken</Operation>
  <URL ref="flow.variable">{value}</URL>
  <GenerateResponse enabled="true">
    <Format>FORM_PARAM | XML</Format>
  </GenerateResponse>
  <GenerateErrorResponse enabled="true">
    <Format>FORM_PARAM | XML</Format>
    <Realm>http://oauth.sap.com/oauth/1/</Realm>
  </GenerateErrorResponse>
</OAuthV1>
```

The policy enforces the following OAuth semantics:

- The consumer key is valid.
- The request token is valid.
- The verifier is valid.
- The signature is valid.

A successful request returns the following responses:

- Generates an access token and its attributes in the flow oauth_token, oauth_token_secret, oauth_response
- Also makes a consumer token and its attributes available in the flow oauth_consumer_key, oauth_consumer_secret

The <URL> element allows proper OAuthV1 signature calculation when running behind Elastic Load Balancers (ELBs). The ref attribute takes precedence when both the ref attribute and a text value are specified. If ref is not provided, or if the flow variable given in ref is NULL, then the text value will be considered for generating the signature. For example:

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<URL ref="secretURL">https://example-{env}.<host:port>/oauth/access_token</URL>
```

On failure, a request returns the appropriate response status code with an error message.

4.2.2.3 Associating a token verification code with a request token

The following policy associates a token verification code with a request token. To receive an access token, both a verification code and a request token are required.

> **Note**
>
> To use this policy, a verification code must be generated as a separate step outside of SAP API Management. The verifier value must be a unique, random string.

The verification code fits into the basic OAuth flow as follows: After an app user's credentials are authenticated by a service provider, the service provider must inform the consumer app that the credentials were accepted, implying that the request token can be exchanged for an access token. This is usually done via a 302 redirect. Separately, the service provider must send a verification code to SAP API Management, and it internally associates this verifier to the request token.

Later, when the app exchanges the request token for an access token, it must pass both the request token and the verifier code. SAP API Management then checks that the verifier code is correct, ensuring that the user who authenticated and granted access to the service is the same user who is asking the app to retrieve an access token. For detailed information on this message flow, see the OAuth 1.0a specification.

4.2.2.4 Comprehensive form

```
<OAuthV1 name="OAuthV1-GenerateVerifier-1">
    <Operation>GenerateVerifier</Operation>
    <RequestToken ref="request.header.requesttoken"/>
    <AppUserId ref="request.header.appuserid"/>
    <VerifierCode ref="request.header.verifiercode"/>
    <GenerateResponse enabled="true">
       <Format>FORM_PARAM | XML</Format>
    </GenerateResponse>
    <Attributes>
       <Attribute name="ver-attr1">ver-attr1</Attribute>
       <Attribute name="ver-attr2" ref="request.header.ver-attr2"></Attribute>
       <Attribute name="ver-attr3" display="false">ver-attr3</Attribute>
    </Attributes>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
</OAuthV1>
```

4.3 TLS / SSL

TLS (Transport Layer Security, whose predecessor is SSL) is the standard security technology for establishing an encrypted link between a web server and a web client, such as a browser or an app. An encrypted link ensures that all data passing between the server and the client remains private. To use SSL, a client makes a secure request to the server by using the encrypted https:// protocol, instead of the unencrypted http:// protocol.

SAP API Management supports one-way SSL and two-way SSL. One-way SSL enables the SSL client to verify the identity of the SSL server. For example, an app running on an Android phone (client) can verify the identity of SAP API Management APIs (server).

SAP API Management also supports a stronger form of authentication using two-way, or client, SSL. You typically implement two-way SSL to enhance security end-to-end and protect your data from client attacks such as client spoofing or man-in-the middle attacks. In two-way SSL, the client verifies the identity of the server followed by the server verifying the identity of the client.

4.3.1 SSL terminology

You should be familiar with some important terms and concepts before configuring SSL:

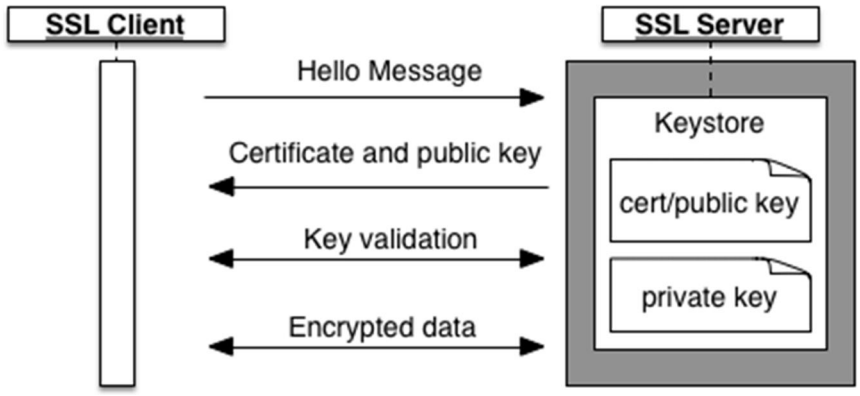
Term	Definition
SSL certificate	A digital file that identifies an entity in an SSL transaction. A certificate, or cert, can be used to identify the SSL server and SSL client, depending on the SSL configuration.
Certificate chain	Often you will not have a certificate signed by your CA's root private key. Instead, you have your cert along with one or more intermediate certs that form a chain. The last intermediate cert in the chain is typically signed by the CA's root private key.
CertificateAuthority(CA)	A trusted entity, such as Symantec or VeriSign, used to issue certs and to validate the authenticity of a cert. One type of cert, called a self- signed cert, does not require a CA.
Public key	Used to encrypt data sent from an SSL client to an SSL server. The public key is included in the cert. All SSL clients have a copy of the server's public key.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Term	Definition
Private key	Used on the SSL server to decrypt data. Only the SSL server has the private key - it is not shared with SSL clients.
Keystore	Contains the SSL certificate and private key used to identify the entity during SSL handshaking.
Truststore	Contains trusted certificates on an SSL client used to validate an SSL server's certificate presented to the client.
Server Name Indication (SNI)	Allows multiple HTTPS targets to be served off the same IP address and port without requiring those targets to use the same certificate.

4.3.1.1 One-way SSL

The following figure shows SSL handshaking for one-way authentication between an SSL client and SSL server.



In a one-way SSL configuration:

- The client issues a session request to the server.
- The server responds with a certificate, which contains its public key. This certificate comes from the server's keystore, which also contains the server's private key. The private key is never sent to the client.
- For a signed cert, the client makes a request to the Certificate Authority (CA) to authenticates the certificate.
- The client and server exchange several more messages to validate keys.
- The client begins SSL data transfer with the server.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

In one-way SSL, SAP API Management can be either the server or the client:

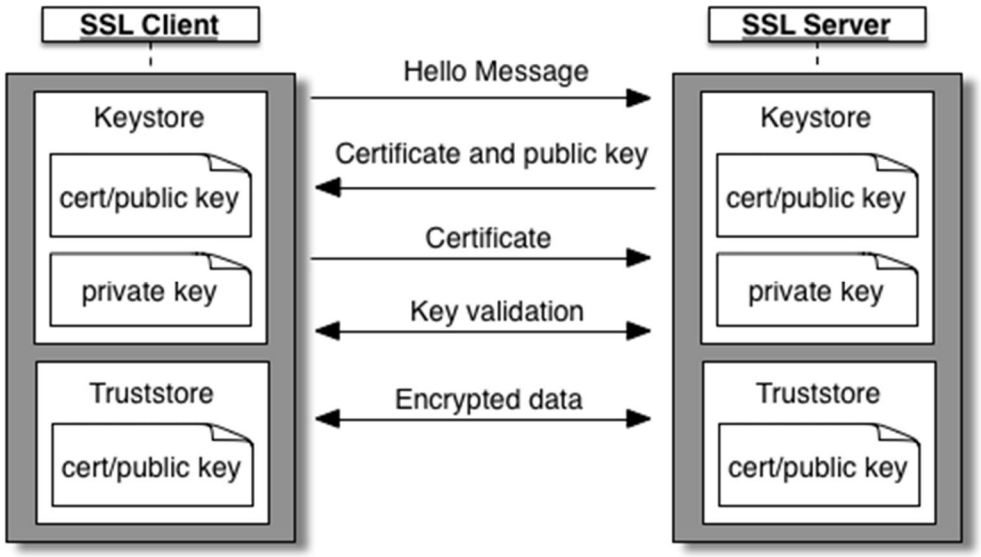
- SAP API Management as the SSL server

  SAP API Management is the server hosting the SSL endpoint, where the SSL endpoint corresponds to an API proxy deployed to a virtual host. The client is an app attempting to access the API proxy. In this scenario, SAP API Management has the keystore containing the certificate and private key.
- SAP API Management as the SSL client

  SAP API Management acts as the client that accesses a backend service. In this case, the backend service corresponds to the server hosting an SSL endpoint. The backend server therefore has a keystore that contains its certificate and private key.

4.3.1.2 Two-way SSL

The following figure shows the SSL handshaking for two-way SSL authentication between a client and server:



In two-way SSL:

- The client and server both have their own keystores. The client's keystore contains its cert and private key, and the server's keystore contains its cert and private key.
- The client has a copy of the server's cert in its truststore. During SSL handshaking, the client compares the cert in its truststore to the cert send from the server to verify the identity of the server.
- The server has a copy of the client's cert in its truststore. During SSL handshaking, the server compares the cert in its truststore to the cert send from the client to verify the identity of the client.
- The SSL server presents its certificate to the SSL client to authenticate itself. The client then verifies the identity of the server prior to sending its cert to the server.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- The SSL client presents its certificate to the SSL server to authenticate itself to the server.

In two-way SSL, SAP API Management can be either the server or the client:

- SAP API Management as the server

  SAP API Management is the server hosting the SSL endpoint, where the SSL endpoint corresponds to an API proxy. The client is an app attempting to access the API proxy. In this scenario, SAP API Management has a keystore containing the certificate and private key, and a truststore containing the client's cert and CA chain.
- SAP API Management as the client

  SAP API Management acts as a client that accesses a backend service. In this case, the backend service corresponds to the server hosting the SSL endpoint. The backend server therefore has a keystore that contains its certificate and private key.

  SAP API Management must also define a keystore that contains the certificate needed to validate itself to the backend service, and a truststore containing the cert from the backend server.

The important thing to remember is that SAP API Management is flexible enough to support two-way SSL regardless of how you decide to configure it.

4.3.1.3 Server Name Indication (SNI) support

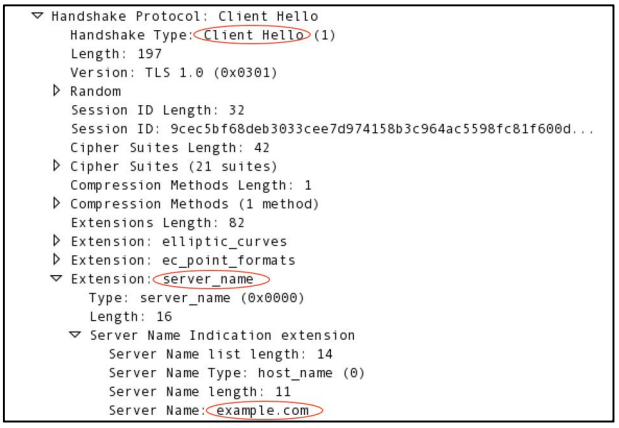
SAP API Managment supports the use of Server Name Indication (SNI) from system to target endpoints. Java 1.6 or 1.7 is required.

With SNI, which is an extension of TLS/SSL, multiple HTTPS targets can be served off the same IP address and port without requiring those targets to use the same certificate.

No system-specific configuration is required. If your target environment is configured for SNI, SAP API Management supports it.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

System automatically extracts the hostname from the request URL and adds it to the SSL handshake request. For example, if the target host ishttps://example.com/request/path, then it adds the server_name extension as shown below:



4.3.1.4 Update a TLS certificate

If an TLS certificate expires, or if your system configuration changes such that the certificate is no longer valid, then you need to update the certificate. The process of updating a certificate depends on your deployment of Edge: cloud or on-premises.

Note:

- You cannot update an existing keystore to add a new certificate. You must create a new keystore when updating a certificate.
- You can optionally choose to delete the existing keystore and then create a new one with the same name. However, for the time from when the certificate expired until you create the new keystore, you cannot service requests.
- If the keystore is used for two-way TLS between SAP API Management Edge and the backend service, and you are using SAP API Management Edge, then after deleting and recreating the keystore with the same name, you must restart the SAP API Management Edge Message Processors.
- If you configured the virtual host or the TargetEndpoint to use a reference to the keystore or truststore, you can update the reference to point to a different keystore or truststore to update the TLS cert. That means Cloud customers do not have to contact SAP API Management Support and Cloud customers do not need to restart a Router or Message Processor. However, Cloud customers must contact SAP API Management Support if they require an update to the virtual host.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.3.1.4.1 Determine when a cert is due to expire

Typically, you create a new keystore before the current certificate expires, and then update your virtual hosts or target endpoints to use the new keystore so that you can continue to service requests without interruption due to an expired certificate. You can then delete the old keystore after ensuring that the new keystore is working correctly.

To check when a certificate is due to expire, go to the Admin > SSL Certificates menu in the SAP API Management Edge UI. You can also configure that page to indicate if a certificate is due to expire in 10, 15, 30, or 90 days.

4.3.1.4.2 Update an TLS certificate in a keystore

The way you update an TLS certificate in a keystore is based on your SAP API Management Edge deployment type: cloud or Private Cloud.

7. Cloud deployment

For a cloud-based deployment of SAP API Management Edge:

1. Create a new keystore as described in KeyStores and TrustStores.
2. Upload a new JAR file containing the new certificate and private key to the keystore.
3. For inbound connections, meaning an API request into SAP API Management Edge, contact SAP Support to update the virtual host configuration to reference the the new keystore and key alias.
4. For outbound connections, meaning from SAP API Management to a backend server:
    1. Update the TargetEndpoint configuration for any API proxies that referenced the old keystore and key alias to reference the new keystore and key alias.

       If your TargetEndpoint references a TargetServer, update the TargetServer definition to reference the new keystore and key alias.
    2. If the keystore and truststore are referenced directly from the TargetEndpoint definition, then you must redeploy the proxy.

       If the TargetEndpoint references a TargetServer definition, and the TargetServer definition references the keystore and truststore, then no proxy redeployment is necessary.
5. After you have confirmed that your new keystore is working correctly, delete the old keystore with the expired cert and key as described above.

4.3.1.4.3 On-premises deployment

For an on-premises deployment of SAP API Management Edge:

1. Create a new keystore as described in Keystores and Truststores.
2. Upload a new JAR file containing the new certificate and private key to the keystore.
3. For inbound connections, meaning an API request into SAP API Management Edge:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

1. Update any virtual hosts that referenced the old keystore and key alias to reference the new keystore and key alias.
2. Restart the Routers, one at a time. Note that if you deleted the old keystore and created a new keystore with the same name, then no Router restart is necessary.

   No proxy redeployment required.

4. For outbound connections, meaning from SAP API Management to a backend server:
   1. Update the TargetEndpoint configuration for any API proxies that referenced the old keystore and key alias to reference the new keystore and key alias.
      If your TargetEndpoint references a TargetServer, update the TargetServer definition to reference the new keystore and key alias.
   2. For any API proxies that reference the keystore and truststore from a TargetEndpoint definition, you must redeploy the proxy.

      If the TargetEndpoint references a TargetServer definition, and the TargetServer definition references the keystore and truststore, then no proxy redeployment is necessary.
   3. If the keystore is used for two-way TLS between SAP API Management Edge and the backend service, and you deleted / recreated the keystore with the same name, you must restart the SAP API Management Edge Message Processors.
5. After you have confirmed that your new keystore is working correctly, delete the old keystore with the expired cert and key as described above.

4.3.1.4.4 Update a TLS certificate in a truststore

The way you update an TLS certificate in a truststore is based on your SAP API Management Edge deployment type: cloud or Private Cloud.

Cloud Deployment

For a cloud-based deployment of SAP API Management Edge:

1. Upload a new cert to the truststore as described in Keystores and Truststores. There is no need to delete the old cert.
2. For both inbound or outbound connections, contact SAP Support.
3. Confirm that your updated truststore is working correctly.

On-premises deployment

For an on-premises deployment of SAP API Management Edge:

1. Upload a new cert to the truststore as described in Keystores and Truststores. There is no need to delete the old cert.
2. For inbound connections, meaning an API request into SAP API Management Edge, restart the Routers, one at a time. No proxy redeployment required.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3. For outbound connections, meaning from SAP API Management to a backend server, restart the SAP API Management Edge Message Processors, one at a time.
4. Confirm that your new truststore is working correctly.

4.3.1.4.5 Update your truststore for an expired SAP API Management cert

SAP API Management provides all Cloud customers with a cert when they create an account so that the customer can get up and running quickly with SAP API Management Edge. All Cloud customers get a copy of the same SAP API Management-provided cert.

Based on when you obtained your cert, the SAP API Management cert will expire on either February 23, 2016 or April 8, 2016. SAP API Management will be updating all keystores before those dates to upload a new cert and private key.

However, if you have uploaded the SAP API Management cert to a truststore, you must add the new SAP API Management cert to your truststore before it expires. You do not have to delete the old SAP API Management cert, you only have to upload the new one.

Shown below is the new cert:

```
-----BEGIN CERTIFICATE-----
MIIFHzCCBAegAwIBAgIISzef6agJ+w4wDQYJKoZIhvcNAQELBQAwgbQxCzAJBgNV
BAYTAlVTMRAwDgYDVQQIEwdBcml6b25hMRMwEQYDVQQHEwpTY290dHNkYWxlMRow
GAYDVQQKExFHb0RhZGR5LmNvbSwgSW5jLjEtMCsGA1UECxMkaHR0cDovL2NlcnRz
LmdvZGFkZHkuY29tL3JlcG9zaXRvcnkvMTMwMQYDVQQDEypHbyBEYWRkeSBTZWN1
cmUgQ2VydGlmaWNhdGUgQXV0aG9yaXR5IC0gRzIwHhcNMTYwMTEzMTcyNTU0WhcN
MTkwNDA5MDYzNzEwWjA6MSEwHwYDVQQLExhEb21haW4gQ29udHJvbCBWYWxpZGF0
ZWQxFTATBgNVBAMMDCouYXBpZ2VlLm5ldDCCASIwDQYJKoZIhvcNAQEBBQADggEP
ADCCAQoCggEBALRAjJBUiGcKTEye40tpQjqGnAYn+LgpfS8Wkhh5nUZFvJBbEVDe
heH4Agc1LILOExEtKj/bGRhhFQzojSfTmz3LuGA8oVnzzOamYDvzkTkK5yO+Yd2C
FXgEOQhphcaZms2Z1Kl9cn1fQRl/+KUbiKgV/piu4079cnPqR8uME94mWnHpRurd
ZJqfOxpz+144RbTNaSJdQdiPQ1vzxmLQFFN2SrbWx4OXni/C5QJ0S14uplZHgw+j
Ae/LGAiaJR7nirot3QEk7qdYRF6g86OvrpToDbG6vKSyuIXoOT33oBJnjfFQRkq+
29VcMWwWLe6lmToGpp0xkNxdeyFB0oyQqFsCAwEAAaOCAawwggGoMAwGA1UdEwEB
/wQCMAAwHQYDVR0lBBYwFAYIKwYBBQUHAwEGCCsGAQUFBwMCMA4GA1UdDwEB/wQE
AwIFoDA3BgNVHR8EMDAuMCygKqAohiZodHRwOi8vY3JsLmdvZGFkZHkuY29tL2dk
aWcyczEtMTc4LmNybDBTBgNVHSAETDBKMEgGC2CGSAGG/W0BBxcBMDkwNwYIKwYB
BQUHAgEWK2h0dHA6Ly9jZXJ0aWZpY2F0ZXMuZ29kYWRkeS5jb20vcmVwb3NpdG9y
eS8wdgYIKwYBBQUHAQEEajBoMCQGCCsGAQUFBzABhhhodHRwOi8vb2NzcC5nb2Rh
ZGR5LmNvbS8wQAYIKwYBBQUHMAKGNGh0dHA6Ly9jZXJ0aWZpY2F0ZXMuZ29kYWRk
eS5jb20vcmVwb3NpdG9yeS9nZGlnMi5jcnQwHwYDVR0jBBgwFoAUQMK9J47MNIMw
ojPX+2yz8LQsgM4wIwYDVR0RBBwwGoIMKi5hcGlnZWUubmV0ggphcGlnZWUubmV0
```

```
MB0GA1UdDgQWBBROfmwEgDjmhQL41zDFmOsQ/Z/ARTANBgkqhkiG9w0BAQsFAAOC
AQEAq0MW3tOUPPdujGc2JLhd3SpYXCoHvps7JkjJ0yHurzXkCmnDHLAfYYXbu7Ei
B+k3anDavgGpO7odJQaoYtbpTCfZHRlUWSaT0Xef3UUIAMuu2bKIaNIYRTKUA7Jj
X4pCVTQCKHuoYMFjTAqm6d+q68WMv2oW6EERjf0irw++Yecuq+CllKIOird2zS73
Fc3RTp1LcM+J0AwHkA3r8KSCQhsqLdn/Y0JTxJ8d2E4RYH2jpkMB1ogA1/VUj9ZW
xTUVrk3Duw4Wq/66mPpQVGZHfRDJ9TBF5Qt8iUKxEIt4IzaWmk70049ygab29XaC
xaC2QIzKJJbBueo76nvKKpfeGQ==
-----END CERTIFICATE-----
```

Copy this cert to a PEM file, and then upload the certificate to your truststore. The way you update your truststore is based on its location and implementation. For example, if it is on your backend servers, use the procedure based on your server implementation.

If you are updating a truststore on SAP API Management Edge, use the Upload a Certificate to a Truststore:

```
$ curl -X POST -H "Content-Type: multipart/form-data" -F file="@newapigeecert.pem" \
https://api.enterprise.apigee.com/v1/o/{org_name}/environments/{env_name}/keystores/myTruststore/certs?alias=myTruststore \
-u email:password
```

where the `-F` option specifies the path to the PEM file.

4.3.2 Using SSL with SAP API Management

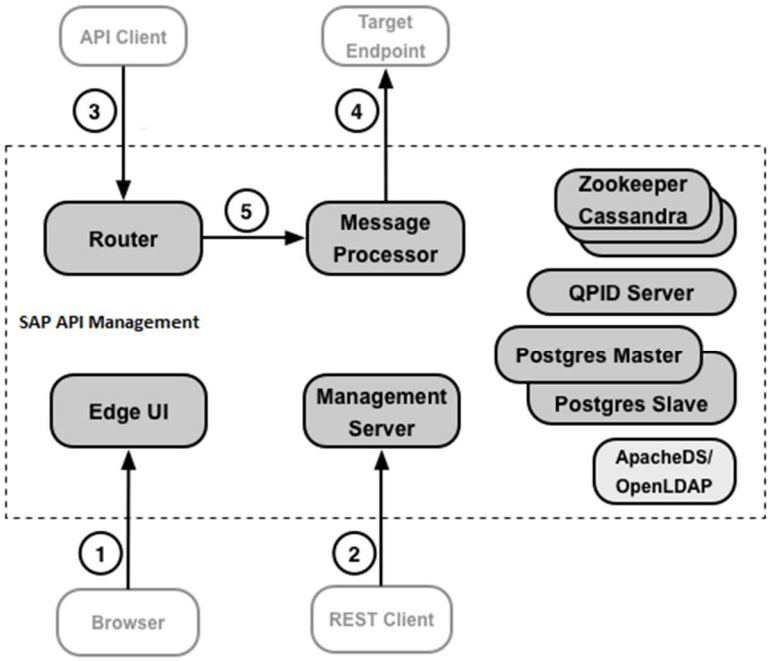
SAP API Management has several entry points that you might want to secure with SSL. In addition, system add-ons, such as the Developer Services portal, have entry points that can be configured to use SSL.

For the Developer Services portal, you are completely responsible for configuring SSL. That means you not only have to obtain the SSL certificate and private key, but you also have to configure SAP API Management to use SSL.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.3.2.1 Where SAP API Management uses SSL

The following images shows the places in an SAP API Management installation where you can configure SSL:



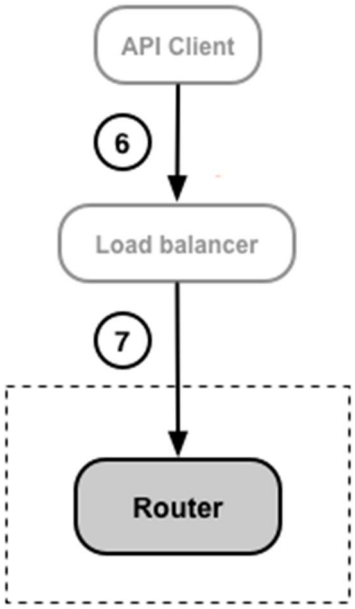
The following table describes these SSL connections:

	Source	Destination	Description
1	API developer	Management UI	The management UI is a browser-based tool that API developers use to perform most of the tasks necessary to create, configure, and manage API proxies and API products.
2	API Developer	management API	All system services can be configured through the management API, a REST-based API. That means you can use these APIs to create, configure, and manage API proxies and API products, create and manage apps and app developers, and to perform many other types of operations.
3	API Client (app)	API	Apps access your APIs by making requests to API proxies through virtual hosts on the SAP API Management Router.
4	SAP API Management	Target endpoint	An API proxy functions as a mapping of a publicly available endpoint on system to a target endpoint, which is often defined by an endpoint on

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

	Source	Destination	Description
			your backend service. The Message Processor accesses your backend service in response to a request to an API proxy.
5	Router	Message Processor	A Router handles all system incoming API traffic, determines the API proxy that handles the request, balances requests across available Message Processors, and dispatches the request.

The following image shows a scenario where the API client accesses SAP API Management through a load balancer, rather than accessing the Router directly:



Note

Load balancers often cannot be configured to support two-way SSL.

When using a load balancer, you can configure SSL between the API client and the load balancer and, if necessary, between the load balancer and the Router, as the following table describes:

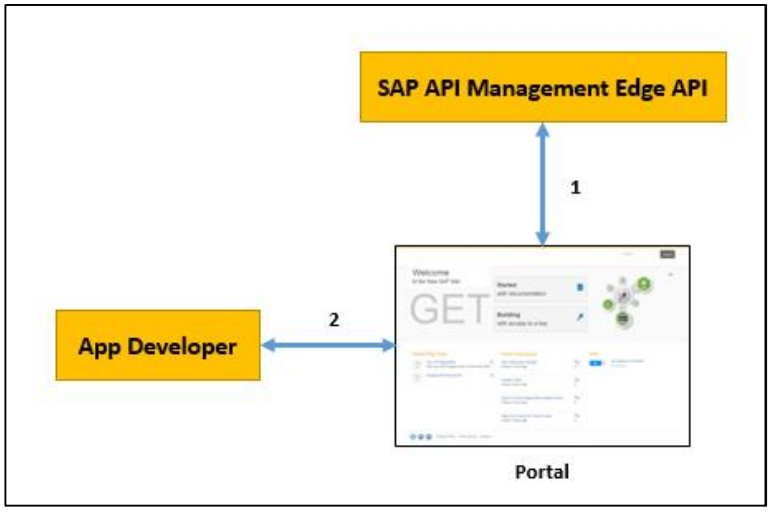
	Source	Destination	Description
6	API Client (app)	Load Balancer	Apps access your APIs by making requests to API proxies through a load balancer. The load balancer forwards the request to an SAP API Management Router.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

	Source	Destination	Description
			You can configure SSL on the entry point of the load balancer. The way you configure SSL is based on the load balancer.
7	Load Balancer	Router	Depending on your configuration, you might configure SSL access to the Router from the load balancer. In that case, you configure SSL just as if the load balancer was not present. Or, if the load balancer and Router are in the same security domain, SSL configuration may not be necessary. However, that is dependent on your network configuration.

4.3.2.2 Where the Developer Services portal uses SSL

The following image show the two places where the portal uses SSL:



© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.3.3 Configuring TLS

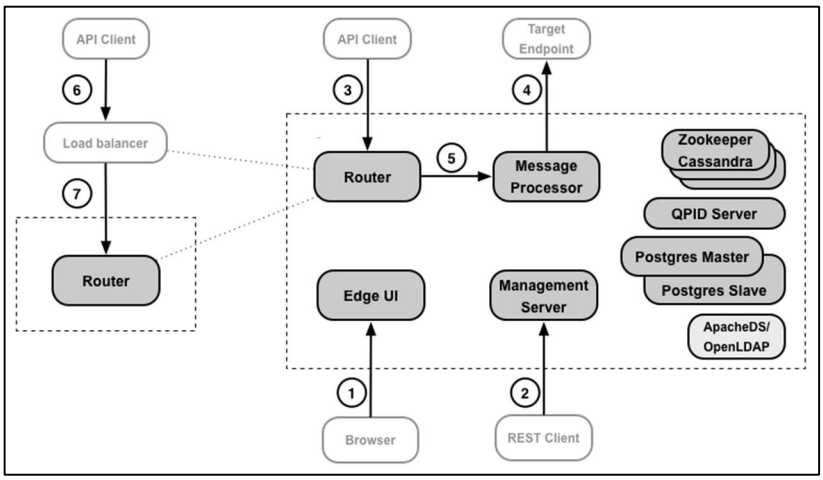
To configure TLS, you have to configure the following on SAP API Management Edge:

- Repositories for TLS keys and certs:
  - Keystores: Contains an TLS certificate and private key used to identify the entity during TLS handshaking. When you create the keystore and upload the TLS cert, you specify an alias name used to reference the cert/key pair.
  - Truststores: contains certificates used to verify certificates received as part of TLS handshaking. A truststore is not required. It is typically used to validate self-signed certificates received from the TLS server, or certificates that are not signed by a trusted CA.

- API proxies to use the certs in keystores and truststores:
  - Virtual Hosts: defines the domains and ports on which an API proxy is exposed, and, by extension, the URL that apps use to access an API proxy. As part of configuring a virtual host, you optionally specify a keystore and truststore as part of configuring TLS.
  - Target Endpoints/Target Servers: defines endpoint of the backend system accessed by an API proxy. As part of configuring a target endpoints/target servers, you configure it to support the TLS requirements of the backend system, including specifying a keystore and truststore

4.3.4 Configuring SSL

The following figure shows where customers configure SSL:



The following table describes the locations where you configure SSL access:

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

	Source	Destination	SSL Access
1	API developer	SAP API Management UI	Enable SSL on the management UI. See the SAP API Management Operations Guide for more.
2	API Developer	SAP API Management API	Enable SSL on the management API. See the Operations Guide for more.
3	API Client (app)	API	Enable SSL on API access through the use of virtual hosts on the Router.
4	SAP API Management	Target endpoint	Enable SSL between SAP API Management and a backend service provider.
5	Router	Message Processor	Enable SSL for communication between a Router and Message Processor. See the Operations Guide for more.

If your installation included a load balancer, then you might also have to replace the configuration for #3 above to configure SSL between the app and load balancer, and between the load balancer and the Router:

	Source	Destination	Description
6	App	Load Balancer	Enable SSL on the load balancer. This process is determined by your load balancer.
7	Load Balancer	Router	If necessary, enable SSL on the Router for requests from the load balancer. Use the same process as you do for configuring SSL for a virtual host. If the load balancer and Router are in the same security domain, SSL configuration may not be necessary. However, that is dependent on your network configuration.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.3.4.1 Creating a virtual host

A virtual host defines the domains and ports on which an API proxy is exposed, and, by extension, the URL that apps use to access an API proxy.

A virtual host also defines whether the API proxy is accessed by using the HTTP protocol, or by the encrypted HTTPS protocol that uses SSL. When configuring a virtual host to use HTTPS and SSL, you create a virtual host and configure the virtual host to use a keystore and truststore.

> **Note**
>
> If your installation uses a load balancer, and the load balancer is responsible for handling encrypted traffic, then you still need to create the virtual host. However, your network configuration will determine whether or not the virtual host has to support SSL between the load balancer and the Router.

4.3.4.1.1 What you need to create a virtual host

Before you create a virtual host, you should have the following information:

The publicly facing domain name of the virtual host. For example, you should know if the publicly facing name is api.myCompany.com,myapi.myCompany.com, etc. That information is used when you create the virtual host and also when you create the DNS record for the virtual host.

- For one-way SSL, you need to create a keystore where the keystore contains the following:
  - SSL certificate as a PEM file - either a certificate signed by a certificate authority (CA), or a chain of certificates where the last certificate is signed by a CA.
  - Private key as a PEM file. SAP API Management supports key sizes up to 2048 bits. A passphrase is optional.
- For two-way SSL, you need a keystore just like you do with one-way SSL and, depending on how you configure two-way SSL, you might also need a client certificate and a truststore to hold that certificate and the certificate's CA chain.

4.3.4.1.2 Virtual host configuration

To create a virtual host, create an XML object that defines the virtual host. For example, the following XML object defines a virtual host that uses the HTTP protocol:

```
<VirtualHost name="myVHost">
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <HostAliases/>
    <Interfaces/>
    <Port>9005</Port>
</VirtualHost>
```

Notice that the virtual host contains a name property. You use the value of the name property to configure an API proxy to use the virtual host.

You can then access an API proxy through this virtual host by making a request to:

```
http://<routerIP>:<port>/{proxy-base-path}/{resource-path}
https://<routerIP>:<port>/{proxy-base-path}/{resource-path}
```

where:

- `http or https:` If the virtual host is configured to supports SSL, use HTTPS. If the virtual host does not support SSL, use HTTP.
- `<routerIP>:<port>` is the IP address and port number of the virtual host.
- `{proxy-base-path}` and `{resource-path}` are defined when you create the API proxy.

Typically, you do not publish your APIs to customers with an IP address and port number. Instead, you define a DNS entry for the Router and port. For example:

```
http://api.myCompany.com/{proxy-base-path}/{resource-path}
https://api.myCompany.com/{proxy-base-path}/{resource-path}
```

If you define the DNS entry, then you must create a host alias for the virtual host that matches the domain name of the DNS entry. From the example above, you would specify a host alias of api.myCompany.com.

```
<VirtualHost name="myVHost">
    <HostAliases>
        <HostAlias>api.myCompany.com:9005</HostAlias>
    </HostAliases>
    <Interfaces/>
    <Port>9005</Port>
</VirtualHost>
```

The next XML object uses the <SSLInfo> element to define a virtual host for a one-way SSL configuration over HTTPS:

```
<VirtualHost name="mySSLVHost">
    <HostAliases>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        <HostAlias>apiSSL.myCompany.com:9006</HostAlias>
    </HostAliases>
    <Interfaces/>
    <Port>9006</Port>
    <SSLInfo>
        <Enabled>true</Enabled>
        <ClientAuthEnabled>false</ClientAuthEnabled>
        <KeyStore>myTestKeystore</KeyStore>
        <KeyAlias>myKeyAlias</KeyAlias>
    </SSLInfo>
</VirtualHost>
```

In this example, the <Enabled> element is set to true to enable one-way SSL, and the <KeyStore> and <KeyAlias> elements specify the keystore and key used by the SSL connection.

To enable two-way SSL, set the <ClientAuthEnabled> element to true.

The value of the <TrustStore> element depends on your configuration. If your configuration requires a truststore, then specify a value for the <TrustStore> element. Otherwise, leave it empty.

The following table lists the XML elements that you use to configure a virtual host:

Element	Description	Default	Required
VirtualHost	Specifies the name of the virtual host. You use that name to reference the virtual host when configuring an API proxy. The characters that you can use in the name attribute are restricted to: A-Z0-9._\-$ %.	None	Yes
Port	Specifies the port number used by the virtual host. Ensure that the port is open on the SAP API Management Router. If you specify a port in a <HostAlias> element, then the port number specified by <Port> must match it. While it is not required, it is a best practice to use a different port for each virtual host. Note: A router can listen to only one HTTPS connection per virtual host, on a specific port, with the specified cert. Therefore, multiple virtual hosts cannot use the same port number if SSL termination occurs on the router at the specified port.	None	Yes
HostAliases			

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Element	Description	Default	Required
HostAlias	The publicly visible DNS name of the virtual host on the Router, optionally including the port number. You must create a DNS record for the virtual host that matches the host alias. While the port number is optional, it is recommended that you specify it. Or, you can specify two<HostAlias> elements, one with the port number and one without. If you specify the port as part of the host alias, you must also specify the same port by using the<Port> element. You can have multiple <HostAlias> definitions in the same <VirtualHost> definition, corresponding to multiple DNS entries for the virtual host, but not multiple ports. If you want multiple ports, create multiple <VirtualHost> definitions with different ports.	None	No
Interfaces			

Interface	Specifies the network interfaces that you want <Port> to be bound to. If you omit this element, the port is bound on all interfaces.
	For example, to specify to bind the port only to en0:
	`<Interfaces>`
	`<Interface>en0</Interface>`
	`</Interfaces>`
	Determine the interfaces available on your system by running the "ifconfig -a" command.
		None	All interfaces

SSLInfo			
Enabled	Enables one-way SSL. You must have defined a keystore containing the cert and private key.	False	no
ClientAuthEnabled	Enables two-way, or client, SSL between SAP API Management (server) and the app (client) making the request. Enabling two-way SSL requires that you set up a truststore on SAP API Management that contains the cert from the SSL client.	False	no
KeyStore	The name of the keystore	None	Yes if Enabled is true
KeyAlias	The alias specified when you uploaded the JAR file containing the cert and private key to the keystore.	None	Yes if Enabled is true

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Element	Description	Default	Required
Truststore	The name of the truststore that contains the certificate or certificate chain used for two-way SSL.	None	No

Ciphers	Specifies the ciphers supported by the virtual host. If no ciphers are specified, then all ciphers available for the JVM will be permitted.
	To restrict ciphers, add the following elements:
	`<Ciphers>`
	`<Cipher>TLS_RSA_WITH_3DES_EDE_CBC_SHA</Cipher>`
	`<Cipher>TLS_RSA_WITH_DES_CBC_SHA</Cipher>`
	`</Ciphers>`
		All supported by the JVM	No

Protocols	Specifies the protocols supported by the virtual host. If no protocols are specified, then all protocols available for the JVM will be permitted.
	To restrict protocols, add the following elements:
	`<Protocols>`
	`<Protocol>TLSv1</Protocol>`
	`<Protocol>SSLv2Hello</Protocol>`
	`</Protocols>`
		All supported by the JVM	No

4.3.4.1.3 Creating a virtual host that uses HTTP

To create a virtual host that uses the HTTP protocol, perform the following:

1. Create the virtual host by using the Create a Virtual Host API, where <ms-IP> is the IP address or domain name of the Management Server node:

```
$ curl -X POST -H "Content-Type:application/xml" \
http://<ms-IP>:8080/v1/o/{org_name}/environments/{env_name}/virtualhosts \
-d '<VirtualHost name="newVHost">
     <HostAliases>
       <HostAlias>api.myCompany.com:9008</HostAlias>
     </HostAliases>
     <Interfaces/>
     <Port>9008</Port>
   </VirtualHost>' \
-u myname:mypass
```

2. Create the DNS record for the virtual host that matches the host alias.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

3. If you have any existing API proxies, add the virtual host to the <HTTPConnection> element in the Proxy Endpoint. The virtual host is added automatically to all new API proxies.

   See Updating an API proxy after creating a virtual host.

   > **Note**
   >
   > All virtual hosts are automatically added to all new API proxies. Therefore, if you create a new API proxy that should not be accessible over a particular virtual host, then you must edit the API proxy to remove that virtual host from its ProxyEndpoint.

After updating an API proxy to use the virtual host, and creating the DNS record for the host alias, access the API proxy as shown below:

```
http://api.myCompany.com/v1/{project-base-path}/{resource-path}
```

For example:

```
http://api.myCompany.com/v1/weather/forecastrss?w=12797282
```

4.3.4.1.4 Creating a virtual host that uses HTTPS

Use the following procedure to create the virtual host:

1. Create and configure the keystore by using the procedure described in KeyStores and TrustStores.

   When you create the keystore, you specify the keystore name and key alias. You need that information to create the virtual host.

2. Create the virtual host by using the Create a Virtual Host API, where <ms-IP> is the IP address or domain name of the Management Server node.

   Make sure to specify the correct keystore and key alias:

```
$ curl -X POST -H "Content-Type:application/xml" \
http://<ms-IP>:8080/v1/o/{org_name}/environments/{env_name}/virtualhosts \
-d '<VirtualHost  name="newSSLTrustStore2">
      <HostAliases>
        <HostAlias>apiSSL.myCompany.com:9005</HostAlias>
      </HostAliases>
      <Interfaces/>
      <Port>9005</Port>
      <SSLInfo>
         <Enabled>true</Enabled>
        <ClientAuthEnabled>false</ClientAuthEnabled>
        <KeyStore>myTestKeystore</KeyStore>
        <KeyAlias>myKeyAlias</KeyAlias>
      </SSLInfo>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  </VirtualHost>' \
-u myname:mypass
```

> **Note**
>
> If you are performing two-way SSL with the client, then set <ClientAuthEnabled> to true. The client must be configured correctly for two-way SSL, meaning SAP API Management has a truststore containing the client's cert and certificate chain.

3. Create a DNS record for the virtual host that matches the host alias.
4. If you have any existing API proxies, add the virtual host to the <HTTPConnection> element in the ProxyEndpoint. The virtual host is added automatically to all new API proxies.

> **Note**
>
> All virtual hosts are automatically added to all new API proxies. Therefore, if you create a new API proxy that should not be accessible over a particular virtual host, then you must edit the API proxy to remove that virtual host from its ProxyEndpoint.

After updating an API proxy to use the virtual host, and creating the DNS record for the host alias, you can access the API proxy as shown below:

```
https://apiSSL.myCompany.com/v1/{project-base-path}/{resource-path}
```

For example:

```
https://apiSSL.myCompany.com/v1/weather/forecastrss?w=12797282
```

4.3.4.1.5 Modifying a virtual host

To modify a virtual host, perform the following:

1. Update the virtual host by using the Update a Virtual Host API, where <ms-IP> is the IP address or domain name of the Management Server node. You must specify the complete definition of the virtual host in the request body, not just the elements that you want to change. In this example, you change the port number of the virtual host from 9008 to 9009:

```
$ curl -X PUT -H "Content-Type:application/xml" \
http://<ms-
IP>:8080/v1/o/{org_name}/environments/{env_name}/virtualhosts/{vhost_name} \
-d '<VirtualHost name="newVHost">
     <HostAliases>
       <HostAlias>api.myCompany.com:9009</HostAlias>
     </HostAliases>
     <Interfaces/>
     <Port>9009</Port>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    </VirtualHost>' \
-u myname:mypass
```

4.3.4.1.6 Deleting a virtual host

To delete a virtual host, perform the following:

1. Delete the virtual host by using the Delete a Virtual Host API, where <ms-IP> is the IP address or domain name of the Management Server node:

```
$ curl -X DELETE \
http://<ms-
IP>:8080/v1/o/{org_name}/environments/{env_name}/virtualhosts/{vhost_name} \
-u myname:mypass
```

4.3.5 Configuring SSL to the Backend Service

In API proxy functions as a mapping of a publicly available endpoint to your backend service. A virtual host defines the way that the public facing API proxy is exposed to an app. For example, the virtual host determines if the API proxy can be accessed by using SSL. When you configure an API proxy, edit its ProxyEndpoint definition to configure the virtual hosts that is uses.

The TargetEndpoint is the outbound equivalent of the ProxyEndpoint. A TargetEndpoint functions as an HTTP client from SAP API Management to a backend service. When creating an API proxy, you can configure it to use zero or more TargetEndpoints.

> **Note**
>
> Node.js: If your proxy target is a Node.js application, you can use the Node.js tls module to create secure connections to backend services. You make outgoing requests with the tls module the same way you would normally in Node.js. Basically, you need to add client-side keys and certificates (.pem files) to the resources/node directory and load them inside your script.

4.3.5.1 Configuring a TargetEndpoint

To configure a TargetEndpoint, edit the XML object that defines the TargetEndpoint. You can edit the TargetEndpoint by editing the XML file that defines the TargetEndpoint in your API proxy, or edit it in the management UI.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

To use the management UI to edit the TargetEndpoint:

1. Login to the management UI.
2. In the management UI menu, select APIs.
3. Select the name of the API proxy to update.
4. Select the Development tab.
5. Under Target Endpoints, select default.
6. In the code area, the TargetEndpoint definition appears, similar to below:

```
<TargetEndpoint name="default">
    <Description/>
    <FaultRules/>
    <Flows/>
    <HTTPTargetConnection>
        <Properties/>
        <URL>https://weather.yahooapis.com</URL>
    </HTTPTargetConnection>
    <PreFlow name="PreFlow">
        <Request/>
        <Response/>
    </PreFlow>
    <PostFlow name="PostFlow">
        <Request/>
        <Response/>
    </PostFlow>
</TargetEndpoint>
```

7. Make any changes and save the proxy. If the API proxy has been deployed, saving it redeploys it with the new setting.

Notice that the TargetEndpoint definition contains a name property. You use the value of the name property to configure the ProxyEndpoint definition of an API proxy to use the TargetEndpoint.

TargetEndpoints can be configured to reference a TargetServer, rather than the explicit target URL.
A TargetServer configuration decouples concrete endpoint URLs from TargetEndpoint configurations.
TargetServers are used to support load balancing and failover across multiple backend server instances.

Shown below is an example TargetServer definition:

```
<TargetServer  name="target1">
  <Host>https://weather.yahooapis.com</Host>
  <Port>80</Port>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  <IsEnabled>true</IsEnabled>
</TargetServer>
```

A TargetServer is referenced by name in the <HTTPTargetConnection> element in a TargetEndpoint definition. You can configure one or more namedTargetServers, as shown below.

```
<TargetEndpoint name="default">
 ...
  <HTTPTargetConnection>
    <LoadBalancer>
      <Server name="target1" />
      <Server name="target2" />
    </LoadBalancer>
    <Path>/test</Path>
  </HTTPTargetConnection>
  ...
</TargetEndpoint>
```

4.3.5.2 Configuring one-way SSL to the backend server

To configure one-way SSL access from SAP API Management (SSL client) to the backend server (SSL server) does not require any additional configuration. It is up to the backend server to configure SSL correctly.

You only need to make sure that the <URL> element in the TargetEndpoint definition, or the <Host> element in a TargetServer definition, references the backend service by the HTTPS protocol.

4.3.5.3 Configuring two-way SSL to the backend server

If you want to support two-way SSL between SAP API Management (SSL client) and the backend server (SSL server):

- Create a keystore and upload the cert and private key. This cert and private key is typically supplied by the backend server.
- Create a truststore that contains the cert and CA chain that you received from the backend server.
- Update the TargetEndpoint of any API proxies that reference the backend server to configure SSL access.

Use the following procedure to configure two-way SSL:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

1. Create the keystore on SAP API Management, and upload the cert and private key, by using the procedure described in Keystores and TrustStores.

   When you create the keystore, you specify the keystore name and key alias. You need that information to configure the TargetEndpoint.
2. Create a truststore , and upload the cert and CA chain, as described in KeyStores and TrustStores.

   When you create the truststore, you specify the truststore name. You need that information to configure the TargetEndpoint.

3. Use the SAP API Management UI to update the TargetEndpoint definition for the API proxy (or, if you define the API proxy in XML, edit the XML files for the proxy):
   1. Login to the management UI.
   2. In the SAP API Management UI menu, select APIs.
   3. Select the name of the API proxy to update.
   4. Select the Development tab.
   5. Under Target Endpoints, select default.
   6. In the code area, edit the <HTTPTargetConnection> element to add the <SSLInfo> element. Make sure to specify the correct keystore and key alias and set both the <Enabled> and <ClientAuthEnabled> elements to true:

```
<TargetEndpoint name="default">
  …
  <HTTPTargetConnection>
    <SSLInfo>
      <Enabled>true</Enabled>
      <ClientAuthEnabled>true</ClientAuthEnabled>
      <KeyStore>myKeystore</KeyStore>
      <KeyAlias>myKey</KeyAlias>
      <TrustStore>myTrustStore</TrustStore>
    </SSLInfo>
    <URL>https://myservice.com</URL>
  </HTTPTargetConnection>
  …
</TargetEndpoint>
```

   Note

   If your TargetEndpoint references a TargetServer instead of a specific URL, update the TargetServer definition to reference the new keystore and key alias. For example:

```
<TargetServer name="target1">
  ...
  <SSLInfo>
    <Enabled>true</Enabled>
    <ClientAuthEnabled>true</ClientAuthEnabled>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        <KeyAlias>myKeystore</KeyAlias>
        <KeyStore>myKey</KeyStore>
      </SSLInfo>
    </TargetServer>
```

You do not have to save the API proxy if you update a TargetServer definition.

7. Save the API proxy. If the API proxy has been deployed, saving it redeploys it with the new setting.

4.3.5.4 Enabling SNI

SAP API Management supports the use of Server Name Indication (SNI) from message processors to target endpoints. Java 1.6 or 1.7 is required.

To be backward compatible with your existing target backends, SAP disabled SNI by default. If your target backend is configured to support SNI, you can enable this feature by setting the following property to `true` in `/<inst-root>/sap4/conf/sap/management-server/system.properties`:

```
jsse.enableSNIExtension=true
```

You must then restart the Message Processors using the command:

```
/<inst-root>/sap4/bin/sap-service message-processor restart
```

4.3.6 KeyStores and TrustStores

To configure functionality that relies on public key infrastructure (SSL and SAML, for example) you need to create keystores and truststores that provide the necessary keys and digital certificates.

4.3.6.1 About keystores and truststores

Keystores and truststores define repositories of security certificates used for SSL encryption. The main difference between the two is where they are used in the SSL handshaking process:

- A keystore contains an SSL certificate and private key.

  In one-way SSL, when a client connects to the SSL endpoint on the server, the server's keystore presents the server's certificate (public cert) to the client. The client then validates that certificate with a certificate authority.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

In two-way SSL, both the client and the server maintain a keystore with their own cert and private key used for mutual authentication.

- A truststore contains trusted certificates stored on the client that are used when the client makes an outbound SSL connection to an SSL endpoint. For two-way SSL, the contents of the truststore are used to validate the identity of the server's certificate being presented to the client. Note that a truststore does not contain a private key.

Certificates can be issued by a certificate authority (CA), or they can be self-signed by the private key that you generate. If you have access to a CA, follow instructions provided by your CA for generating keys and issuing certificates. If you do not have access to a CA, you can generate a self-signed certificate using one of the many publicly available free tools, such as openssl.

Implementing a keystore and truststore on SAP API Management

On SAP API Management, a keystore contains one or more JAR file, where the JAR file contains a:

- SSL certificate as a PEM file - either a certificate signed by a certificate authority (CA), a chain of certificates where the last certificate is signed by a CA, or a self-signed sert.
- Private key as a PEM file. SAP API Management supports key sizes up to 2048 bits. A passphrase is optional.

A truststore is similar to a keystore except that it contains only certs, as a PEM files, but no private keys. If the cert is part of a chain, then the truststore must contain all certs in the chain as individual PEM files.

SAP API Management provides an API that you use to create keystores and truststores. The actual APIs are the same. The difference is that when you create a keystore, you pass a JAR file that contains the cert and private key. When you create a truststore, you pass only the cert as a PEM file.

About the format of the cert and key files

The examples in this document show the SSL cert and key defined as PEM files, which comply with the X.509 format. If your cert or private key is not defined by a PEM file, you can convert it to a PEM file by using utilities such as openssl.

However, many .crt files and .key files are already in the PEM format. If these files are text files, and are enclosed in:

```
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
```

or:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
 -----END ENCRYPTED PRIVATE KEY-----
```

Then the files are compatible with the PEM format and you can use them in a keystore or truststore without converting them to a PEM file.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

If you have a certificate chain, and want to use that chain in a keystore, then all of the certs have to be combined into a single PEM file. The certs have to be in order and the last cert must be a root certificate or an intermediate cert signed by a root certificate:

```
-----BEGIN CERTIFICATE-----
(Your Primary SSL certificate)
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
(Intermediate certificate)
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
(Root certificate or intermediate certificate signed by a root certificate)
-----END CERTIFICATE-----
```

> **Note**
>
> When using a truststore, if you have a chain of certs, you must upload the chain as individual PEM files.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.3.6.2 Get details about an existing keystore

Check your environment for any existing keystores by using the List Keystores and Truststores API:

```
$ curl -X GET \
https://<host:port>/v1/o/{org_name}/environments/{env_name}/keystores \
-u email:password
```

The returned array is empty until you create your first keystore.

Check the contents of the keystore by using the Get a Keystore or Truststore API.

You can also view this information in the SAP API Management UI:

1. Login to the management UI http://<ms-ip>:9000 , where <ms-ip> is the IP address of the Management Server node.
2. In the management UI menu, select Admin > SSL Certificates.

4.3.6.3 Create a JAR file containing your cert and private key

Create a JAR file with your private key, certificate, and a manifest. The JAR file must contain the following files and directories:

```
/META-INF/descriptor.properties
myCert.pem
myKey.pem
```

> **Note**
>
> A keystore JAR can contain only one certificate. If you have a certificate chain, all certs in the chain must be appended into a single PEM file, where the last certificate is signed by a CA. The certs must be appended to the PEM file in the correct order, meaning:
>
> cert -> intermediate cert(1) -> intermediate cert(2) -> ... -> root

In the directory containing your key pair and certificate, create a directory called /META-INF. Then, create a file called descriptor.properties in /META-INF with the following contents:

```
certFile={myCertificate}.pem
keyFile={myKey}.pem
```

Generate the JAR file containing your key pair and certificate:

```
$ jar -cf myKeystore.jar myCert.pem myKey.pem
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Add descriptor.properties to your JAR file:

```
$ jar -uf myKeystore.jar META-INF/descriptor.properties
```

4.3.6.4 Create a keystore

A keystore is specific to an environment in your organization, for example the test or prod environment. Therefore, if you want to test the keystore in a test environment before deploying it to your production environment, you must create it in both environments.

To create a keystore in an environment, you only need to specify the keystore name to the Create a Keystore or Truststore API:

```
$ curl -H "Content-Type: text/xml" \
https://<host:port>/v1/o/{org_name}/environments/{env_name}/keystores \
-d '<KeyStore name="myKeystore"/>' -u email:password
```

Sample response:

```
{
 "certs" : [ ],
 "keys" : [ ],
 "name" : "myKeystore"
}
```

After you create a named keystore in an environment, you can upload your JAR files that contain a cert and private key by using the Upload a JAR file to a Keystore API:

```
$ curl -X POST -H "Content-Type: multipart/form-data" \
-F file="@myKeystore.jar" \
"https://<host:port>/v1/o/{org_name}/environments/{env_name}/keystores/{myKeystore}/keys?alias={key_alias}&password={key_pass}" \
-u email:password
```

where the -F option specifies the path to the JAR file.

In this call, you specify two query parameters:

- `alias` - Identifies the certificate and key in the key store. When you create a virtual host, you reference the certificate and key by its alias name.
- `password` - The password for the private key. Omit this parameter if the private key has no password.

Verify that your keystore uploaded properly:

```
$ curl
https://<host:port>/v1/o/{org_name}/environments/{env_name}/keystores/myKeystore \
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
-u email:password
```

Sample response:

```
{
 "certs" : [ "myCertificate" ],
 "keys" : [ "myKey" ],
 "name" : "myKeystore"
}
```

4.3.6.5 Create a truststore

The APIs that you use to create a truststore are the same as used to create a keystore. The only difference is that you pass the cert file as a PEM file instead of a JAR file.

> **Note**
>
> You often create a truststore on SAP API Management as part of configuring two-way SSL between SAP API Management and a backend server. In this case, when you configure the truststore on SAP API Management you upload a cert file that you received from the entity hosting the backend server.

If the cert is part of a chain, then you must upload all certs in the chain separately to the truststore. The final certificate is typically signed by the certificate issuer. For example, in the truststore, you upload a client certificate, client_cert_1, and the client certificate issuer's certificate, ca_cert.

During two-way SSL authentication, client authentication succeeds when the server sends client_cert_1 to the client as part of the SSL handshaking process.

Alternatively, you have a second cert, client_cert_2, signed by the same cert, ca_cert. However, you do not upload client_cert_2 to the truststore. The truststore still contains client_cert_1 and ca_cert.

When the server passes client_cert_2 as part of SSL handshaking, the request succeeds. This is because SAP API Management allows SSL verification to succeed when client_cert_2 does not exist in the truststore but was signed by the a cert that exists in the truststore. If you remove the CA certificate,ca_cert, from the truststore then SSL verification fails.

Create an empty truststore in the environment by using Create a Keystore or Truststore, the same API that you use to create a keystore:

```
$ curl -X POST -H "Content-Type: text/xml" -d \
'<KeyStore name="myTruststore"/>' \
https://<host:port>/v1/o/{org_name}/environments/{env_name}/keystores \
-u email:password
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Upload the certificate as a PEM file to the truststore by using the Upload a Certificate to a Truststore API:

```
$ curl -X POST -H "Content-Type: multipart/form-data" -F file="@trust.pem" \
https://<host:port>/v1/o/{org_name}/environments/{env_name}/keystores/myTruststore/certs?alias=myTruststore \
-u email:password
```

where the -F option specifies the path to the PEM file.

4.3.6.6 Delete a keystore or truststore

You can delete a keystore or truststore by using the Delete a Keystore or Truststore API:

```
$ curl -X DELETE \
https://<host:port>/v1/o/{org_name}/environments/{env_name}/keystores/myKeystoreName \
-u email:password
```

Sample response:

```
{
 "certs" : [ ],
 "keys" : [ ],
 "name" : "myKeystoreName"
}
```

If you delete and recreate a keystore or truststore that is being used by a virtual host, then you must redeploy your API proxies. See Understanding deployment.

4.3.6.7 Get SSL certificate details

You can use the Get a Cert from a Keystore or Truststore API to view details about SSL certificates in the keystore, such as expiration date and issuer. First, obtain the name of the certificate in which you are interested. This example fetches information for the keystore called "freetrial".

```
$ curl https://<host:port>/v1/o/{org_name}/environments/{env_name}/keystores/freetrial \
-u email:password
```

Sample response:

```
{
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
 "certs" : [ "wildcard.sap.net.crt" ],
 "keys" : [ "freetrial" ],
 "name" : "freetrial"
}
```

Then, use the value of the certs property to get the certificate details:

```
$ curl
https://<host:port>/v1/o/{org_name}/environments/{env_name}/keystores/freetrial/certs/
wildcard.sap.net.crt \
-u email:password
```

Sample response:

```
{
 "certInfo" : [ {
   "expiryDate" : "Wed, 23 Apr 2014 20:50:02 UTC",
   "isValid" : "Yes",
   "issuer" : "CN=Go Daddy Secure Certificate Authority - G2,
OU=http://certs.godaddy.com/repository/, O=&quot;GoDaddy.com, Inc.&quot;,
L=Scottsdale, ST=Arizona, C=US",
   "subject" : CN=*.example.sap.net, OU=Domain Control Validated",
   "subjectAlternativeNames" : ["*.example.sap.net","*.example.sap.net" ],
   "validFrom" : "Tue, 15 Apr 2014 09:17:03 UTC",
   "version" : 3
 } ],
 "name" : "example.sap.net.crt"
}
```

You can also view this information in the SAP API Management UI:

1. Login to the management UI at http://<ms-ip>:9000 , where <ms-ip> is the IP address of the Management Server node.
2. In the SAP API Management UI menu, select Admin > SSL Certificates.

In the SAP API Management UI, you can specify how far in advance SAP API Management indicates that a certificate is going to expire. By default, the UI highlights any certificates scheduled to expire in the next 10 days.

4.3.6.8 Update an expired certificate

If an SSL certificate expires, then you need to update the certificate. The process of updating a certificate depends on your deployment of SAP API Management: cloud or on-premises.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Note

- You cannot update an existing keystore to add a new certificate. You must create a new keystore when updating a certificate.

- You can optionally choose to delete the existing keystore and then create a new one with the same name. However, for the time from when the certificate expired until you create the new keystore, you cannot service requests.

  Typically, you create a new keystore before the current certificate expires, and then update your virtual hosts or target endpoints to use the new keystore so that you can continue to service requests without interruption due to an expired certificate. You can then delete the old keystore after ensuring that the new keystore is working correctly.

  To check when a certificate is due to expire, go to the Admin > SSL Certificates menu in the management UI. You can also configure that page to indicate if a certificate is due to expire in 10, 15, 30, or 90 days.

1. Create a new keystore as described above.
2. Upload a new JAR file containing the new certificate and private key to the keystore.
3. For inbound connections, meaning an API request into SAP API Management:
   1. Update any virtual hosts that referenced the old keystore and key alias to reference the new keystore and key alias.
   2. Restart the routers, one at a time. Note that if you deleted the old keystore and created a new keystore with the same name, then no Router restart is necessary.

      No proxy redeployment required.
4. For outbound connections, meaning from SAP API Management to a backend server:
   1. Update the TargetEndpoint configuration for any API proxies that referenced the old keystore and key alias to reference the new keystore and key alias.
      If your TargetEndpoint references a TargetServer, update the TargetServer definition to reference the new keystore and key alias.
   2. For any API proxies that reference the keystore and truststore from a TargetEndpoint definition, you must redeploy the proxy.

      If the TargetEndpoint references a TargetServer definition, and the TargetServer definition references the keystore and truststore, then no proxy redeployment is necessary.
5. After you have confirmed that your new keystore is working correctly, delete the old keystore with the expired cert and key as described above.

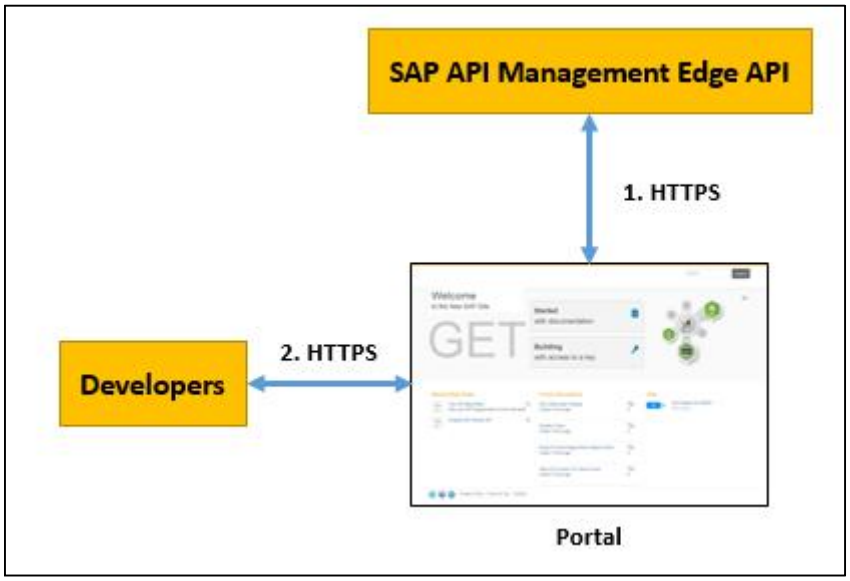
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.3.7 Using SSL On Developer Portal

SSL (Secure Sockets Layer) is the standard security technology for establishing an encrypted link between a web server and a web client, such as a browser or app. An encrypted link ensures that all data passing between the web server and the client remains private. To use SSL, a client makes a secure request to the web server by using the secure https:// protocol, instead of the insecure http:// protocol.

4.3.7.1 SSL and the portal

The following image show the two places where the portal uses SSL:



1. For communication between the portal and the SAP API Management Edge API.

   The portal does not function as a stand-alone system. Instead, much of the information used by the portal is actually stored on SAP API Management. When necessary, the portal makes an HTTP or HTTPS request to the management API to retrieve information or to send information.

   When you create your portal, one of the first steps you must perform is to specify the URL of the management API. Depending on how the management API is configured, that URL can use SSL.
2. For communication between developers and the portal.

   When you use the Developer Services portal to deploy your APIs, your developers log in to the portal to register apps and receive API keys. The login credentials and the API key are proprietary information that you want to send over HTTPS to ensure their security. This type of proprietary information should be sent over HTTPS.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

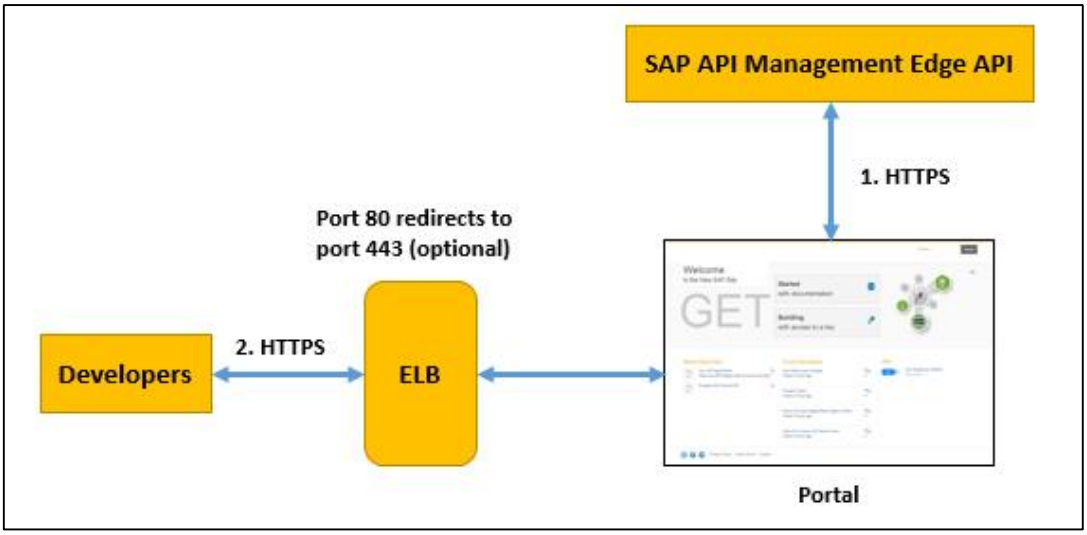
The URL of the management API is in the form

`https://ManagementServerIP:SSLport/v1`

where ManagementServerIP is the IP address of the SAP API Management Server server and SSLport is the SSL port for the management API. For example, the port number could be 8443 or even 8080 based on the configuration.

4.3.7.2 Configuring SSL between developers and the portal

The portal require the portal to be behind a load balancer, as shown below.



Therefore, you have two options for configuring SSL:

- Configure SSL on the load balancer: Configure SSL on the load balancer itself, and not on the portal. The procedure that you use to configure SSL is therefore dependent on the load balancer. See the documentation on your load balancer for more information.
- Configure SSL on the portal itself: If necessary, you can configure SSL on the web server that hosts the portal. By default, SAP API Management installs the Apache web server. For information on configuring SSL for Apache, see https://www.drupal.org/https-information.

Additional SSL settings for settings.php

You can edit the sites/default/settings.php file to make configuration changes to SSL for the portal. When editing the sites/default/settings.php file, add instances of the ini_set() function to set a property. For more information on this function, see:http://php.net/manual/en/function.ini-set.php.

You can set the following properties in the sites/default/settings.php file:

- cookie_httponly: (Recommended) Specifies that cookie as accessible only over the HTTP protocol. Set this property as:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

ini_set('session.cookie_httponly', true);

- session.cookie_secure - (Optional) Specifies that cookies can only be sent over secure connections. However, this means that all content must be served over HTTPS. If this setting is enabled, the site will not work over HTTP. Set this property as:

  ini_set('session.cookie_secure', true);

- gc_maslifetime and cookie_lifetime: (Optional) gc_lifeteime specifies the number of seconds after which data can potentially be cleaned up, andcookie_lifetime specifies the lifetime of the cookie in seconds. Set these properties as:

  ini_set('session.gc_maxlifetime', 3600);
  ini_set('session.cookie_lifetime', 3600);

4.3.7.3 Configuring SSL with Load Balancers

For better performance, load balancers are sometimes configured to perform SSL termination. With SSL termination, load balancers decrypt messages sent over https://and forward the messages to backend servers over http://. That saves backend servers the overhead of decrypting https://messages themselves.

If load balancers forward unencrypted http messages to servers in the same data center, security is not an issue. However, if load balancers forward messages over http://to servers outside the data center, such as your SAP API Management developer portal, the messages are unencrypted, which opens a security hole.

If your developer portal sits behind load balancers that are using SSL termination, and you want all traffic served over https://, the website pages will need to contain https://links only and you will need to add the following code to your developer portal sites/default/settings.php file. Because the load balancer does not automatically transform the content of the HTML pages, the code ensures that all links passed to the client start with https://.

To configure SSL with load balancers, add the following lines to the sites/default/settings.php file:

```
$_SERVER['HTTPS'] = 'on';
$base_url = "https://myurl.com";
$can_detect_ssl = FALSE;

if (isset($_SERVER['HTTP_X_FORWARDED_PROTO']) || (isset($_SERVER['HTTPS']) &&
$_SERVER['HTTPS'] == 'on')) {
 $can_detect_ssl = TRUE;
}
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
if (isset($_SERVER['HTTP_X_FORWARDED_PROTO']) &&
strtolower($_SERVER['HTTP_X_FORWARDED_PROTO']) == 'https') {
 $_SERVER['HTTPS'] = 'on';
}

if ($can_detect_ssl && $_SERVER['HTTPS'] != 'on') {
 header('Location: https://' . $_SERVER['SERVER_NAME'] . $_SERVER['REQUEST_URI']);
 exit;
}
```

4.4 API Keys

API keys provide a simple mechanism for authenticating apps. SAP API Platform generates API keys for apps, and enables you to add API key-based authentication to your APIs using policies. Using SAP API Platform, you can issue, administer and validate API keys without writing any code. This topic shows you how to add API key validation to an API. To learn about API key management, start with the Developer Services

> **Note**
>
> API keys go by many names. You may see them referred to as 'API keys', 'app keys', and 'consumer keys'. All of these names are synonymous.

When apps are registered with SAP API Management Developer Services, a consumer key and secret pair is generated for the app. SAP API Management API Platform stores the consumer key for future validation. The app developer embeds the consumer key in the client app. The client app must present the consumer key for each request. API Platform validates the consumer key before permitting the app's request.

API key validation is the simplest form of app-based security that you can configure for an API. Apps simply present an API key, and SAP API Platform checks to see that the API key is in an approved state for the resource being requested. For this reason the security associated with API keys is limited. API keys can easily be extracted from app code and used to access an API. You may find that API keys work better as unique app identifiers than as security tokens.

4.4.1 Policy Configuration

You can set up API key validation for an API by attaching a policy of type ValidateAPIKey. The only required setting for a ValidateAPIKey policy is the expected location of the API key in the client request. The API proxy will check the location that you specify, and extract the API key. If the API key is not present in the expected location, then an error is thrown and the request is rejected. API keys can be located in a query parameter, a form parameter, or an HTTP header.

For example, the policy configuration below defines the expected key location as a query parameter named apikey. A successful request must present the API key as a query parameter appended to the request, for example,`?apikey=Y7yeiuhcbKJHD790`.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

To validate API keys, create the following policy:

```
<VerifyAPIKey name="APIKeyValidation">
  <APIKey>request.queryparameter.apikey</APIKey>
</VerifyAPIKey>
```

This policy can be attached to any API that you need to protect.

API proxies automatically passthrough all HTTP headers and query parameters that are present on the request. Therefore, after the API key has been validated, it's a good idea to strip it from the message so that the API key is not sent over the wire to the backend service. You can do that using a policy of type AssignMessage as follows:

```
<AssignMessage name="StripApiKey">
    <DisplayName>Remove Query Param</DisplayName>
    <Remove>
        <QueryParams>
            <QueryParam name="apikey"/>
        </QueryParams>
    </Remove>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
    <AssignTo createNew="false" transport="http" type="request"></AssignTo>
</AssignMessage>
```

4.4.2 Policy Attachment

The policies must be attached to an API proxy Flow as processing Steps. By applying the policy to the request PreFlow, API keys are verified on every request received by the API proxy from a client app. After verification, the API key is stripped from the outbound request.

Attach the policies to the ProxyEndpoint of the API proxy to be protected as follows:

```
<ProxyEndpoint name="default">
  <PreFlow>
    <Request>
      <Step><Name>ValidateAPIKey</Name></Step>
      <Step><Name>StripApiKey</Name></Step>
    </Request>
  </PreFlow>
```

After you attach the policy, deploy the API proxy.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.4.3 Submitting a Request with a valid API Key

As an admin in your organization, you can retrieve any app's API key as follows:

```
$ curl http://<host:port>/v1/o/{myorg}/developers/{developer_email}/apps/{app_name}
-u myname:mypass
```

> **Note**
>
> Remember to substitute your organization for {myorg} and your username and password for myname:mypass.

The app profile that is returned for this call provides the consumer key (API key) and secret. The consumer key value is the value you use for the API key in your request to the protected API.

For example, a request that does not include an API key results in an authorization failure.

```
$ curl http://<host:port>/weather/forecastrss?w=12797282
```

The failure message indicates that the policy checked for an API key but did not find a valid key:

```
OAuth Failure : Could not resolve the app key with variable
request.queryparam.apikey
```

When the consumer key for the app is included as a query parameter, the expected result is successful authorization:

```
$ curl http://{org_name}-
test.<host:port>.net/weather/forecastrss?w=12797282&"apikey=PulSCqMnXGchW0pC0s5o9ng
HVTWMeLqk"
```

The expected result is a successful response from the weather service.

Modifying the value of the API key value in the request results in an authorization failure:

```
$ curl http://<host:port>/weather?forecastrss?w=12797282&"apikey=PulSCqMnXGchW0"
```

Results in:

```
OAuth Failure : Consumer Key is Invalid
```

Remember, as an admin for your organization, you can retrieve the consumer key for any app registered in an organization:

```
$ curl http://<host:port>/v1/o/{myorg}/developers/{developer_email}/apps/{app_name}
-u myname:mypass
```

4.5 SAML

The Security Assertion Markup Language (SAML) specification defines formats and protocols that enable applications to exchange XML-formatted information for authentication and authorization.

SAP API Management API Services enables you to authenticate and authorize apps that are capable of presenting SAML tokens. A SAML token is a digitally signed fragment of XML that presents a set of "assertions". These assertions can be used to enforce authentication and authorization.

To use SAML terminology, API Services can function as a service provider (SP) or an Identity Provider (IP). When API Services validates SAML tokens on inbound requests from apps, it acts in the role of SP. (API Services can

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

also act in the IP role, when generating SAML tokens to be used when communicating with backend services. See Last-mile security).

The SAML policy type enables API proxies to validate SAML assertions that are attached to inbound SOAP requests. The SAML policy validates incoming messages that contain a digitally-signed SAML assertion, rejects them if they are invalid, and sets variables that allow additional policies, or the backend services itself, to further validate the information in the assertion.

To validate SAML tokens, you need to make digital certificates available to the SAML policy by creating at least one TrustStore. TrustStores are scoped to environments in your organizations. Thus, you can configure different trust chains in test and prod, ensuring that test SAML tokens cannot be used in prod, and vice-versa.

For details on SAML validation, See Authenticate and authorize using SAML 2.0.

4.6 Last-Mile Security

Last-mile security protects the backend services that are proxied by API Platform. The primary goal of last-mile security is to prevent so-called "end-run" attacks, where an app developer discovers the URL for a backend service and bypasses any API proxies to directly hit the backend URL.

Following are the primary options for setting up last-mile security:

- Client SSL
  or
- Outbound authentication

4.6.1 Client SSL

The primary mechanism for securing the last-mile is client SSL, which is also known as 'mutual authentication'.

See Client-SSL to backend servers.

4.6.2 Outbound Authentication

Last-mile security can also be enforced by requiring the API proxy to present a credential to the backend service.

For example, you may wish to have an API proxy present an API key to your backend service. You could also have an API proxy obtain and present an OAuth client credentials access token.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.6.3 API key

API keys can be applied to outbound requests from API proxies to backend services. This assumes that the backend service is an API that is capable of issuing and validating API keys.

If you do set up an API proxy to present an API key on outbound requests, you must store the API key in a place where it can be retrieved by the API proxy at runtime. One location available for storing API keys is a key/value map. See Persist data using KeyValueMap.

You can use the AssignMessage policy type to add the API key as an HTTP header, query parameter, or payload element to the outbound request. See Generate or modify messages using AssignMessage.

> **Note**
>
> To avoid exposing API keys over network, always configure server-side SSL on your backend services for outbound transactions that use API keys.

4.6.4 OAuth Client Credentials

OAuth client credentials can be used to add a layer of revocability to API keys. If your backend services support OAuth client credentials, you can configure an API proxy to present a client credentials access token for each request.

The API proxy must be configured to perform a callout to obtain the access token from your token endpoint. The API proxy is also required to cache the access token, to prevent it from obtaining a new access token for each call.

> **Note**
>
> Your backend services must be capable of issuing and validating access tokens using the client credentials grant type for this to work.

> **Recommendation**
>
> Always configure server-side SSL on your backend services for transactions that use access tokens.

A number of approaches can be used to implement outbound client credentials.

You can modify this sample to call your token endpoint to obtain an access token. This sample uses JavaScript to attach the token to the outbound request as an HTTP Authorization header. You could also use AssignMessage for this purpose.

4.7 Content-Based Security

Message content is a significant attack vector used by malicious API consumers. API Platform provides a set of Policy types to mitigate the potential for your backend services to be compromised by attackers or by malformed request payloads.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.7.1 JSON threat protection

JSON attacks attempt to use structures that overwhelm JSON parsers to crash a service and induce application-level denial-of-service attacks.

Such attacks can be mitigated using the JSONThreatProtection Policy type.

See Minimize vulnerabilities using JSONThreatProtection.

4.7.2 XML threat protection

XML attacks attempt to use structures that overwhelm XML parsers to crash a service and induce application-level denial-of-service attacks.

Such attacks can be mitigated using the XMLThreatProtection Policy type.

See Minimize API Vulnerabilities Using XML Threat Protection .

4.7.3 General content protection

Some content-based attacks use specific constructs in HTTP headers, query parameters, or payload content to attempt to execute code. An example is SQL-injection attacks. Such attacks can be mitigated using the RegularExpressionProtection Policy type.

See Evaluate message content using RegularExpressionProtection.

4.8 Data Masking

4.8.1 Overview

SAP API Management enables developers to capture message content to enable runtime debugging of APIs calls. In many cases, API traffic contains sensitive data, such credit cards or personally identifiable health information (PHI) that needs to filtered out of the captured message content.

To meet this requirement, SAP API Management defines 'mask configurations' that enable you to specify data that will be filtered out of trace sessions. Masking configurations can be set globally (at the organization-level) or locally (at the API proxy level). Role-based capabilities govern which users have access to the data that is defined as sensitive.

> **Note**
>
> Data masking is only enabled when a trace session (also called a 'debug' session) is enabled for an API proxy. If no trace session is enabled on an API proxy, then the data will not be masked.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.8.2 Using Mask Configurations

Mask configurations enable you to identify sensitive data in these sources:

- XML payloads: Using XPath, you identify XML elements to be filtered from request or response message payloads.
- JSON payloads: Using JSONPath, you identify JSON properties to be filtered from request or response message payloads.
- Flow variables: You can specify a list of variables that should be masked in debug output.

The basic structure of a mask configuration is shown by the following XML representation:

> **Note**
>
> The name of the mask must be *default*.

```
<MaskDataConfiguration name="default">
  <XPathsRequest>
    <XPathRequest>/sap:Greeting/sap:User</XPathRequest>
  </XPathsRequest>
  <XPathsResponse>
    <XPathResponse>/sap:Greeting/sap:User</XPathResponse>
  </XPathsResponse>
  <JSONPathsRequest>
    <JSONPathRequest>$.store.book[*].author</JSONPathRequest>
  </JSONPathsRequest>
  <JSONPathsResponse>
    <JSONPathResponse>$.store.book[*].author</JSONPathResponse>
  </JSONPathsResponse>
  <XPathsFault>
    <XPathFault>/sap:Greeting/sap:User</XPathFault>
  </XPathsFault>
  <JSONPathsFault>
    <JSONPathFault>$.store.book[*].author</JSONPathFault>
  </JSONPathsFault>
  <Variables>
    <Variable>request.header.user-agent</Variable>
    <Variable>request.formparam.password</Variable>
  </Variables>
</MaskDataConfiguration>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

4.8.3 Configuring a mask configuration resource

Configure a Maskconfig resource using the following elements.

Recommendation

If you use ServiceCallout to make a request, the information in that request is not masked with the normal Maskconfig configuration. If you wish to mask ServiceCallout request information, add flow to the variable `ServiceCallout.request` to the **`<Variables>`** element of the Maskconfig configuration.

Field Name	Description	Default	Required?
XPathsRequest	A list of XPath expressions that will be evaluated against XML payloads (if any) in the request path. Any XPaths that successfully resolve will result in the value of the XML element being masked.	N/A	No
XPathsResponse	A list of XPath expressions that will be evaluated against XML payloads (if any) in the response path. Any XPaths that successfully resolve will result in the value of the XML element being masked.	N/A	No
JSONPathsRequest	A list of JSONPath expressions that will be evaluated against JSON payloads (if any) in the request path. Any JSONPaths that successfully resolve will result in the value of the JSON property being masked.	N/A	No
JSONPathsResponse	A list of JSONPath expressions that will be evaluated against JSON payloads (if any) in the response path. Any JSONPaths that successfully resolve will result in the value of the JSON property being masked.	N/A	No
XPathsFault	A list of XPath expressions that will be evaluated against XML payloads (if any) in the error flow (which executes if a fault is thrown at any point in the flow). Any XPaths that successfully resolve will result in the value of the XML element being masked.	N/A	No
JSONPathsFault	A list of JSON expressions that will be evaluated against XML payloads (if any) in the error flow (which executes if a fault is thrown at any point in the flow). Any JSONPaths that successfully resolve will result in the value of the XML element being masked.	N/A	No
Variables	A list of variables (either pre-defined or custom) who values will be masked. For a list of default variables, see Variables reference.	N/A	No
JSONPathsFault	A list of JSON expressions that will be evaluated against XML payloads (if any) in the error flow (which executes if a fault is thrown at any point in the flow).	N/A	No

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name	Description	Default	Required?
	Any JSONPaths that successfully resolve will result in the value of the XML element being masked.		
Variables	A list of variables (either pre-defined or custom) who values will be masked. For a list of default variables, see Variables reference.	N/A	No

4.8.4 Mask Configuration API

Mask configurations are defined as XML- or JSON-formatted files that you upload and download using the RESTful management API.

To see existing mask configurations, you can simply call the API resource /maskconfigs in your organization:

```
$ curl http://<host:port>/v1/o/{org_name}/maskconfigs \
-u myemail:mypass
```

To see mask configurations defined for specific API proxies, you can call the /maskconfigs API:

```
$ curl http://<host:port>/v1/o/{org_name}/apis/{api_name}/maskconfigs \
-u myemail:mypass
```

To see a specific mask configuration, specify the name of the mask:

```
$ curl http://<host:port>/v1/o/{org_name}/maskconfigs\default \
-u myemail:mypass
-----------------------------------------------
$ curl http://<host:port>/v1/o/{org_name}/apis/{api_name}/maskconfigs\default \
-u myemail:mypass
```

To create a mask configuration, use the POST verb to submit a payload that defines the mask configuration:

```
$ curl -H "Content-type:text/xml" -X POST -d \
'<MaskDataConfiguration name="default">
  <XPathsRequest>
    <XPathRequest>/sap:Greeting/sap:User</XPathRequest>
  </XPathsRequest>
  <XPathsResponse>
    <XPathResponse>/sap:Greeting/sap:User</XPathResponse>
  </XPathsResponse>
  <JSONPathsRequest>
    <JSONPathRequest>$.store.book[*].author</JSONPathRequest>
  </JSONPathsRequest>
  <JSONPathsResponse>
    <JSONPathResponse>$.store.book[*].author</JSONPathResponse>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    </JSONPathsResponse>
    <XPathsFault>
      <XPathFault>/sap:Greeting/sap:User</XPathFault>
    </XPathsFault>
    <JSONPathsFault>
      <JSONPathFault>$.store.book[*].author</JSONPathFault>
    </JSONPathsFault>
    <Variables>
      <Variable>request.header.user-agent</Variable>
       <Variable>request.formparam.password</Variable>
    </Variables>
  </MaskDataConfiguration>' \
  http://<host:port>/v1/o/{org_name}/maskconfigs \
  -u email:password
```

To create a mask configuration that is scoped to a specific API proxy:

```
  $ curl -H "Content-type:text/xml" -X POST -d \
  '<MaskDataConfiguration name="default">
    <XPathsRequest>
      <XPathRequest>/sap:Greeting/sap:User</XPathRequest>
    </XPathsRequest>
    <XPathsResponse>
       <XPathResponse>/sap:Greeting/sap:User</XPathResponse>
    </XPathsResponse>
    <JSONPathsRequest>
       <JSONPathRequest>$.store.book[*].author</JSONPathRequest>
    </JSONPathsRequest>
    <JSONPathsResponse>
      <JSONPathResponse>$.store.book[*].author</JSONPathResponse>
    </JSONPathsResponse>
    <XPathsFault>
      <XPathFault>/sap:Greeting/sap:User</XPathFault>
    </XPathsFault>
    <JSONPathsFault>
      <JSONPathFault>$.store.book[*].author</JSONPathFault>
    </JSONPathsFault>
    <Variables>
      <Variable>request.header.user-agent</Variable>
       <Variable>request.formparam.password</Variable>
    </Variables>
  </MaskDataConfiguration>' \
  http://<host:port>/v1/o/{org_name}/apis/{api_name}/maskconfigs \
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
-u email:password
```

You can delete a mask configuration using the DELETE verb:

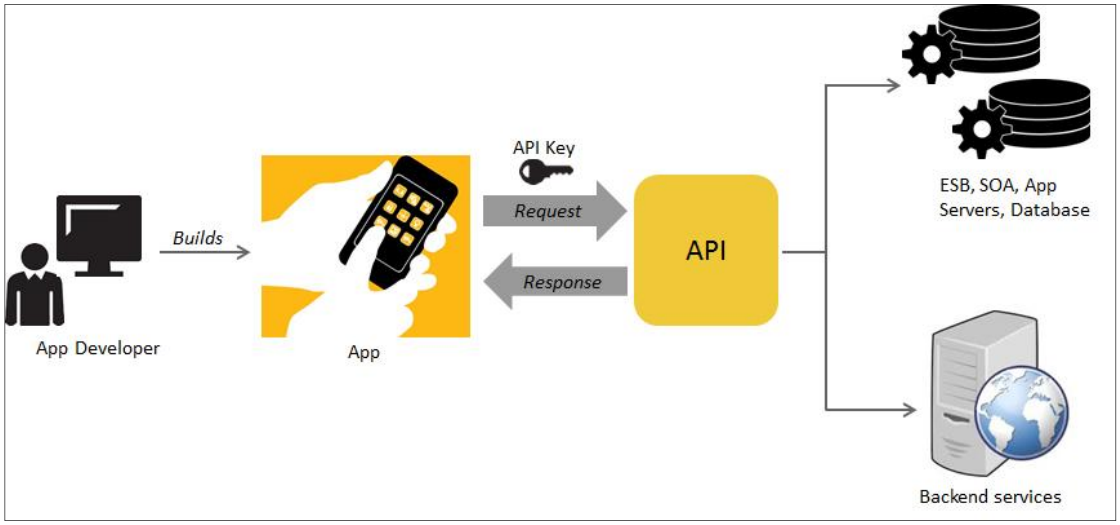
```
$ curl -X DELETE \
http://<host:port>/v1/o/{org_name}/apis/{api_name}/maskconfigs/{maskconfig_name} \
-u email:password
```

The response to a DELETE operation is an HTTP code 204 with no message content.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

5 Publishing APIs

A developer builds an app that makes requests to your APIs to access your backend services. The following image shows the relationship of a developer to an app and to an API:



Publishing is the process of making your APIs available to app developers for consumption. SAP API Management supports a flexible publishing model that gives you complete control over the publishing process.

5.1 Publishing Overview

Publishing APIs can be broadly defined by the following tasks:

1. Create the API products on SAP API Management that bundle your APIs.
2. Register app developers on SAP API Management.
3. Register apps on SAP API Management.

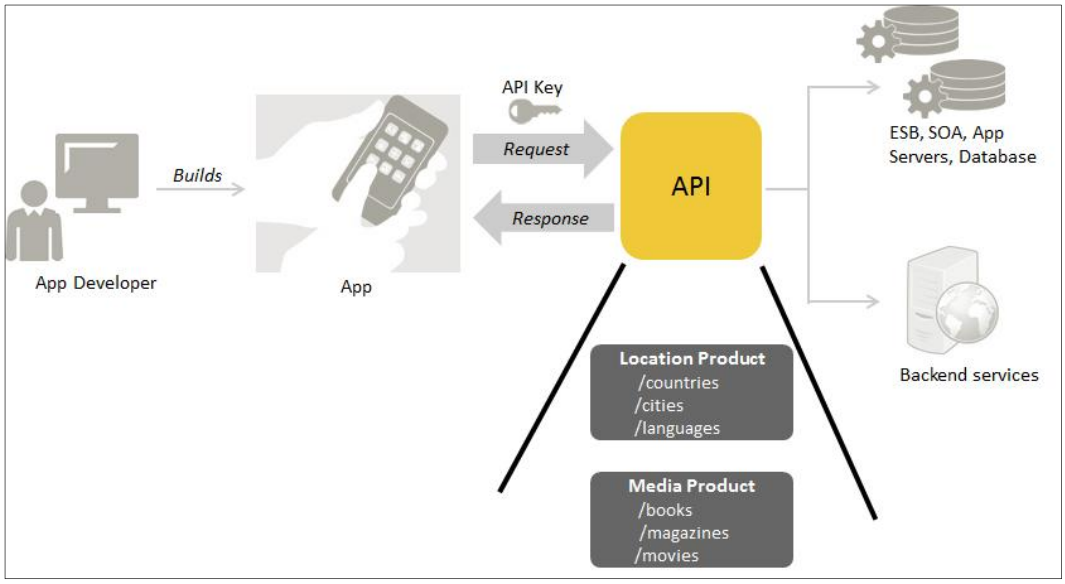
Provide documentation and community support for your APIs, including:

- API reference documentation
- Examples and tutorials
- Forums, blogs, and other features to foster the developer community

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

5.1.1 Task 1: Create an API product

The first task in publishing is to create an API product. An API product is a collection of API resources that are offered as a package to app developers for consumption. Create API products by using the SAP API Management API or UI.



In this figure, your API consists of two products, each containing three API resources.

As an API provider, you are responsible for building the APIs and API products to handle access control, usage restrictions, and any other business requirements. For example, you might:

- Release a free API product that allows read-only access to its API resources.
- Release a second API product for a low price that allows read/write access to the same API resources as the free version but with a low access limit, such as 1000 requests per day.
- Release a third API product for a higher price that allows read/write access the the same API resource but with a high access limit.

The important thing to remember is that SAP API Management gives you the flexibility to create API products that match the business requirements of your APIs.

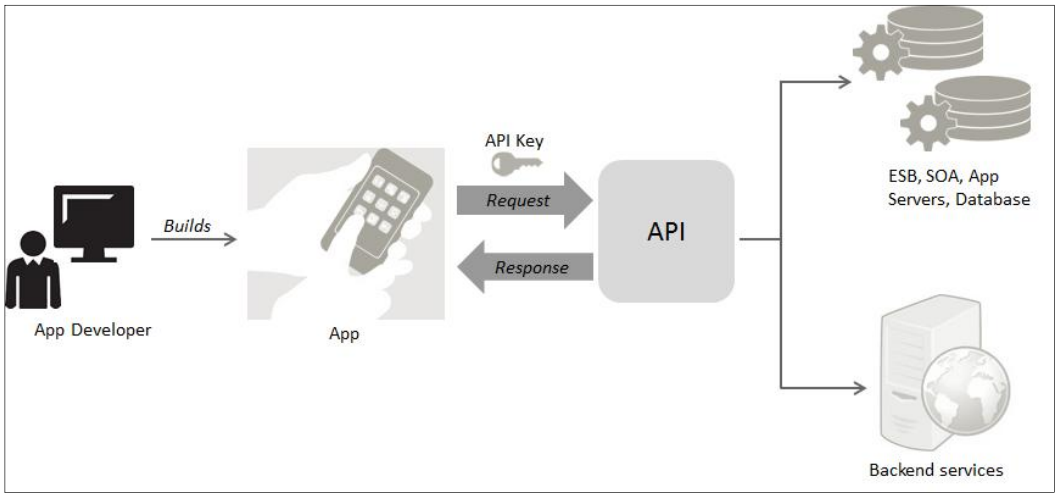
For more information on creating API products, see Creating API products.

5.1.2 Task 2: Register an App Developer

A developer creates the apps that consume your APIs. Developers must be registered on SAP API Platform before they can register an app and receive an API key.

Through the app registration process, you control who has access to your APIs. At any time, you can delete an app developer, which invalidates all API keys associated with that developer, therefore denying that developer access to your APIs.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.



As an API provider, you decide how to register developers. For example, you can use a manual registration process that requires a potential developer to contact your organization to register. The potential developer must supply all necessary information, such as an email address, first and last name, and company name. If you approve the developer's request, you can use the SAP API Management UI to manually register the developer. See Adding developers to your organization for more.

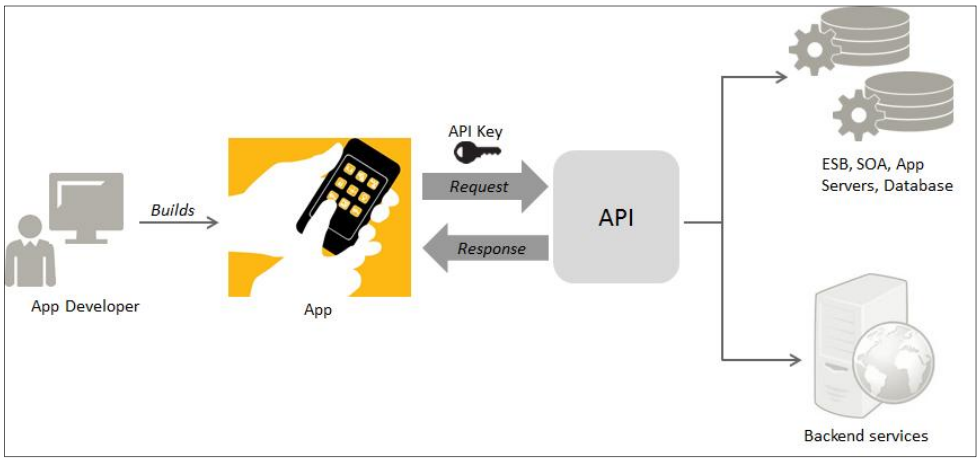
SAP API Management also provides tools that you can use to automate the developer registration process. For example:

Use the SAP API Management API to integrate registration functionality into your existing website. The SAP API Management API is a REST API that you can use to perform all aspects of the developer registration process.

Use the SAP API Management Developer Services portal to register developers. The portal has built-in support for developer registration, but also has many other features to support your APIs. For more details, refer the Developer Portal Guide.

5.1.3 Task 3: Register an App

Before an app can access your APIs, the app must be registered on SAP API Management. However, only a registered developer can register an app on SAP API Management.



© 2014 SAP SE or an SAP affiliate company. All rights reserved.

At the time of app registration, the developer selects one or more API products. For example, you might publish multiple API products corresponding to different types of services and pricing plans. The app developer can then pick and choose from the list of available API products.

> **Note**
>
> SAP API Management does not require that you to let the developer associate multiple API products with an app. Your publishing plan can be structured such that the developer can select only a single API product. Or, you can automatically assign a default API product to all apps so that the developer does not even get to choose one.

In response to registering the app on SAP API Management, it assigns a unique API key to the app. The app must pass that API key as part of every request to an API resource. The key is authenticated and, if valid, the request is granted. At any time, you as the service provider can revoke the key so that the app can no longer access your APIs.

As an API provider, you decide how you want to register apps. You could:

- Use a manual process that requires a developer to contact your organization to register their app. In response, you would send the developer the API key, possibly by email.
- Use the SAP API Management API to integrate app registration functionality and key delivery into your website.
- Use the SAP API Management Developer Services portal which has built in support for app registration and API key delivery.

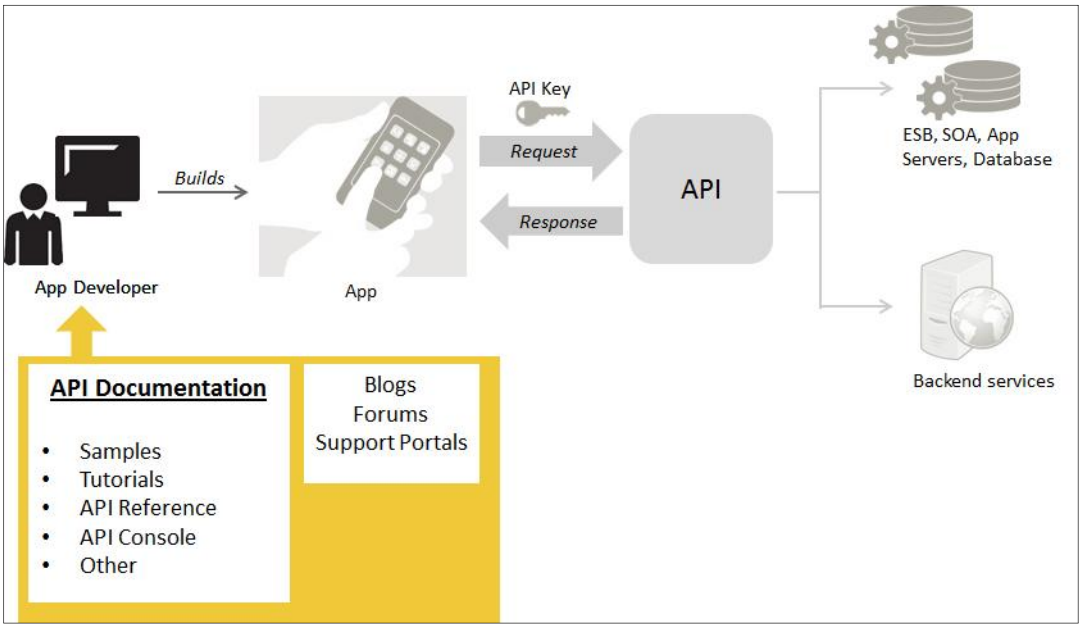
> **Note**
>
> As a best practice, there is a 1-1 correlation between the app that the developer registers on SAP API Management and the actual client app that they build. However, this correlation is not enforced by SAP API Management. For example, a developer registers a single app on SAP API Management, associated that app with an API product, and receives back an API key. The developer is free to use the API key from any client app.

For more information, see Registering apps.

5.1.4 Task 4: Document your APIs

An important consideration for publishing API products is providing documentation and a developer feedback mechanism. Developer portals with social publishing features are increasingly being used for communication with the development community. This includes communicating static content, such as API documentation and terms-of-use, as well as dynamic community-contributed content such as blogs and forums, as well as customer support features.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.



You can build your own website to deploy your documentation. The portal has built-in support for documentation, blogs, forums, and other types of content required to support your developer community.

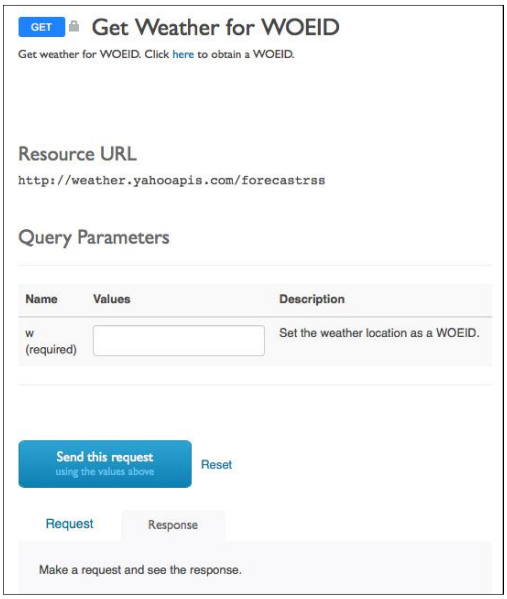
5.1.5 SmartDocs

SmartDocs lets you document your APIs on the Developer Services portal in a way that makes the API documentation fully interactive. Interactive documentation with SmartDocs means portal users can:

- Read about the API
- Send a live request to the API
- View a live response returned from the API

For example, the following figure shows an API documented on the portal by using SmartDocs. This API provides weather information for a specific location:

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.



The developer enters a value for the 'w' query parameter to specify the location, and then chooses the Send the request button to see the live request and response. By creating an interactive documentation on your APIs, you make it easy for portal user to learn, test, and evaluate your APIs.

The SAP API Management management API is a REST API that enables you to access API Platform using any HTTP client. SAP API Management uses SmartDocs to create interactive documentation for the SAP API Creating API Products, Developers and Apps

5.1.6 How to document your APIs by using SmartDocs

SmartDocs represents your APIs by using a model, where the model contains all the information about your APIs. The portal extracts information about your APIs from the model to render the documentation pages on the portal as Drupal nodes, where each Drupal node corresponds to a page of documentation on the portal.

The general steps that you follow to use SmartDocs are:

1. Configure the Drupal SmartDocs module on the portal.
2. Create a SmartDocs model.
3. Add APIs to the model from a WADL file, Swagger definition, or manually.
4. Render the model as a collection of Drupal nodes. Each Drupal node contains information about a single API. For example, if a resource in your API supports both a POST and a PUT request, SmartDocs creates a separate Drupal node for the POST and PUT.
5. Publish the Drupal nodes. Once published, your portal users can view and interact with your API.
6. Edit the Drupal nodes, either before or after you publish them. You can edit the Drupal nodes by using the Drupal editor or by editing the original WADL file or Swagger definition. When you are done editing the WADL file or Swagger definition, import it back into the model as a new revision, then render and publish your changes.
7. Enable SSL. Because SmartDocs can send authentication credentials to your backend as part of making a request to your APIs, you should enable SSL on your portal to ensure that those credentials are secure. In the portal production and test environments, SAP API Management provides the SSL certificate required to make

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

https:// requests. However, before you go live with your portal, you must obtain your own SSL certificate and enable SSL.

5.1.7 Configuring the SmartDocs module

SAP API Management implemented SmartDocs as a custom Drupal module. Use the following procedure to configure the SmartDocs module.

Note

If you are connecting the portal to SAP API Management for the Private Cloud, then you must ensure that SmartDocs was installed on the Edge Management Server at the time you installed SAP API Management Edge. For more information, see the "Install and Configuration Guide" in the documentation for the SAP API Management for the Private Cloud.

To configure the SmartDocs module:

1. Log in to your portal as a user with admin or content creation privileges.
2. Select Modules in the Drupal administration menu. The list of all installed Drupal modules appears.
3. Enable the SmartDocs module.
4. Save the configuration.
5. Select Modules in the Drupal admin menu.
6. Select SmartDocs -> Permissions and ensure that "Perform administration tasks for the SmartDocs module" for "Administrator" role is enabled.
7. Select Configuration > Dev Portal Settings in the Drupal administration menu.
8. Set the Connection Timeout and Request Timeout to 16 seconds.
9. Save the configuration.
10. Configure URL settings:
    1. Select Configuration > Search and metadata > URL aliases > Settings from the Drupal menu.
    2. Set the Maximum alias length and Maximum component length to 255.
    3. Expand Punctuation.
    4. For the Left curly bracket ({) and Right curly bracket (}) settings, select No action (do not replace).
    5. Save the configuration.
11. If your portal connects to an on-premises installation of SAP API Management for the Private Cloud, and the IP address of your Management Server is not publicly accessible:
    1. Run the following cURL command from a machine that can access the Management Server:
       `curl http://managment_server_ip:8080/v1/o/smartdocs/apimodels/proxyUrl`

       where managment_server_ip is the IP address or your Management Server. The response will be in the form:

```
{
    "authUrl": "https://<managment_server_ip:8080>/v1/users/{user}/authenticate",
    "proxyUrl": "https://apiconsole-prod.sap.net/smartdocs/v1/sendrequest"
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

}

2. Copy the URL from the proxyUrl element in the response.
3. Select Configuration > SmartDocs in the Drupal administration menu.
4. Expand the Advanced Settings area.
5. Set the SAP API Management server is firewalled checkbox.
6. Paste the proxyUrl into Send Request Proxy.
7. Save your changes.

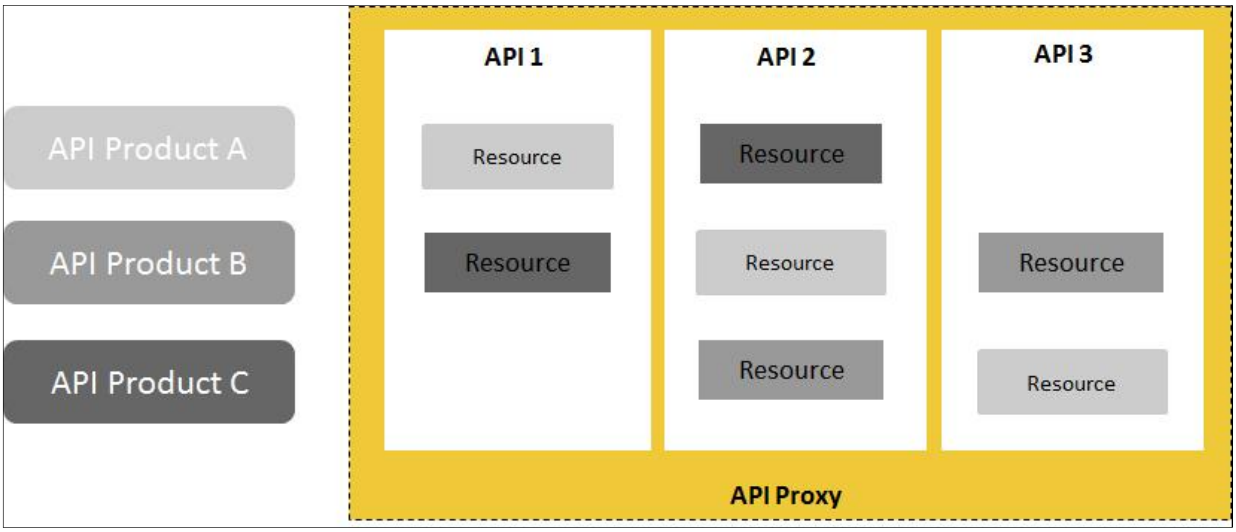
5.1.8 What is an API product?

An API product is a collection of API resources (URIs) combined with a service plan and presented to developers as a bundle. The API product can also include some metadata specific to your business for monitoring or analytics. Refer What is an API and Understanding API proxies for more details.

You can think of API products as your product line. You can create different products to provide features for different use cases. So instead of just giving developers a list of resources, you can bundle specific resources together to create a product that solves a specific user need. For instance, you can create a product that bundles a number of mapping resources to let developers easily add maps to their applications.

API products are also a good way to control access to a specific bundle of resources. For example, you can bundle resources that can only be accessed by internal developers, or bundle resources that can only be accessed by paying customers.

The API resources bundled in a product can come from one or more APIs, so you can mix and match resources to create specialized feature sets.



You can set different properties on each API product. For example, you might make available one API product with a low access limit, such as 1000 requests per day, for a bargain price. You then release another API product that provides access to the same resources, but with a much higher access limit, for a higher price. Or, you might

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

create a free API product that allows read-only access to resources, and then sell an API product to the same resources that allows read/write access.

API products are the central mechanism for authorization and access control to your APIs. In SAP API Management, API keys are provisioned, not for APIs themselves, but for API products. In other words, API keys are provisioned for bundles of resources with an attached service plan. When an app attempts to access an API product, authorization is enforced by SAP API Management at runtime to ensure that:

- The requesting app is permitted to access a particular API resource.
- The requesting app has not exceeded the permitted quota.
- If defined, the OAuth scopes defined in the API product matches those associated with the access token presented by the app.

To learn how to create API products, see Creating API products

5.1.8.1 Access API products from an App

Apps are how developers access your API products. When a developer registers an app, they select the API products to associate with the app, and SAP API Management generates an API key for the app. By default, a single key provides access to all API products associated with the app. When the app makes a request, SAP API Management first verifies that the API key is valid and fails the request if not.

To learn how to create apps, see Registering apps

5.1.8.2 Register developers for your API

Developers build the apps that access your APIs. However, a developer must first be registered in your organization before they can register an app.

To learn how to register developers, see Adding developers to your organization.

5.1.9 Creating API Products

Create an API product in the SAP API Management UI. You must create your API products using the SAP API Management before you can make them available on your developer portal.

> **Recommendation**
>
> You can set up an API product with no resources to make it easier to get up and running quickly. Just create a single API product and provide the base path along with a wildcard for the resource. The wildcard will be interpreted by the system at runtime as meaning that any requested resource in the URI tree below the wildcard is permitted. See the procedure below for more information on using a wildcard in an API product definition.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

5.1.9.1 Before you begin

This section explains a few key concepts related to API products. It's helpful to familiarize yourself with these concepts before you create a new API product.

5.1.9.1.1 Keys

When you register a developer's app in your organization, the app must be associated with at least one API product. As a result of pairing an app with one or more API products, SAP API Management assigns the app a unique consumer key. When the app makes a request to an API resource in an API product, the request must include its unique API key.

API key enforcement doesn't happen automatically. You must enforce API key checking in your API proxies with a policy, such as the Verify API Key policy. Without some type of key enforcement policy, anyone can make calls to your APIs. For more information, see Enforce access control using VerifyAPIKey.

5.1.9.1.2 Manual key approval

By default, all key requests to an API product from an app are automatically approved. You can instead choose to approve keys manually. If you set this option in the SAP API Management UI when creating the product, you will have to approve key requests that come in from any app that adds the API product. For more information, see Registering Apps

5.1.9.1.3 Quotas

You can control the traffic flow for each API product by setting up a quota. Quotas can protect your backend servers for high traffic, and differentiate your product line. For example, you might want to bundle resources with a high quota as a premium product and use the same bundle with a lower quota as a basic product. A quota can help protect your servers from being overwhelmed if a product is particularly popular. See Rate limit API traffic using Quota for more.

5.1.9.1.4 Scope

As an added level of security, you can define any OAuth scopes that must be present in access tokens sent through the product. When you're creating a product, you need to be aware of all the scopes your organization uses. The scopes you add to a product must match existing scopes or the product is not secure.

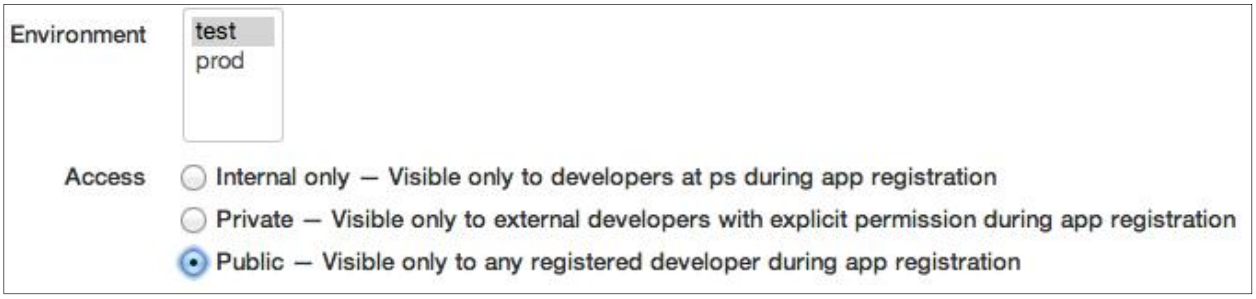
For more information about using scopes with OAuth policies, see Authorize requests using OAuth 2.0.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

5.1.9.2 Creating an API product

To create a new API product:

1. Login to the SAP API Management UI.
2. Choose the Publish tab, then Products
3. Choose the (+) Product button.
4. On the Add Products page, enter a Display Name and Description for the API product.
5. Select the test environment for internal-facing products or the prod environment for public-facing products.
6. Specify an Access level.
   These options determine who can access the product. You can use these levels to control access at different stages of development.
   Only products marked Public are available to developers in the SAP API Management developer portal.
   For example, you can set a product to 'Internal Only' while it's in development and then change access to Public when it's ready to release.



7. Select a Key Approval Type as Automatic or Manual.
   If you select automatic key approval, all key requests that come in from any app that uses this API product are automatically approved. If you select manual key approval, you will have to approve key requests that come in from any app that uses this API product.
8. Enter a Quota limit and select a time period (week, hour, minute, second).
   This sets up a quota for your product that limits the number of calls the product accepts in a given time period.
9. Optionally, if you are using OAuth with the API product, enter the Allowed OAuth Scopes for the product (such as 'Read').
   The OAuth scope should match one of the scopes you defined in your security policy. If they don't match your API may not be secure.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

10. In the Resources section:

    Add an API proxy and resource. Select a proxy version, and a Resource Path, and then select Import Resource. You can select a specific path, or you can select all subpaths with a wildcard.

    > **Note**
    >
    > Wildcards (/** and /*) are supported. The double asterisk wildcard indicates that all sub-URIs are included. A single asterisk indicates that only URIs one level down are included.

    - Add an explicit resource path by selecting the +Custom Resource button.
    - Add an API proxy by selecting the +API Proxy button.

11. Save your product.

5.1.9.2.1 Configuring an API proxy Added to a Product

An app developer chooses the API product to access when the developer registers the app. The developer then receives a key that the app must pass on every request to a resource defined by the product.

However, enforcement of the key is performed at the API proxy level, not by the API product itself. Therefore, you must ensure that all API proxies, and the corresponding resources defined by those API proxies, implement some form of key validation. For example, you can define an API proxy to perform key validation by using the VerifyAPIKey policy. Or, an API proxy could use OAuth 2.0 as defined by the OAuth v2.0 policy. For more, see API keys and OAuth.

5.1.9.2.2 API proxy resources validation

When you have API proxy resource files (such as JavaScript or Java JARs) stored at the environment or organization scope, the validation framework no longer requires you to also include those resources at the API proxy level in a proxy bundle for import to pass validation. Resource validation now occurs at deploy time, not at import time.

5.1.9.2.3 Configure the timeout for individual API proxies

You can configure API proxies to time out after a specified time (with a 504 gateway timeout status). The primary use case is for Private Cloud customers who have API proxies that take longer to execute than the timeout configured on the load balancer, router, and message processor. For example, if the message processor times out first at 2 minutes, you can configure an API proxy to time out after 3 minutes. The following API proxy configuration overrides the SAP API Management Edge system timeout with the specified value in milliseconds for only this proxy:

```
<ProxyEndpoint name="default">
  <HTTPProxyConnection>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <BasePath>/v1/weather</BasePath>
    <Properties>
<!-- api.timeout is in milliseconeds -->
      <Property name="api.timeout">360000</Property>
    </Properties>
  ...
```

Customers, who can't modify the SAP API Management Edge timeouts, can also configure an API proxy timeout, as long as the timeout is shorter than the standard SAP API Management Edge message processor timeout of 57 seconds.

You cannot populate the value with a variable.

5.1.9.2.4 Deleting Resources

You can delete resources that you've added to an API product. You might want to do this if a resource is malfunctioning or requires more development. When deleted, that resource is no longer part of the API product. Any app that uses the API product can no longer access the deleted resource. Deleted resources are removed from the product but are not deleted from the system, so they can still be used by other products.

5.1.9.2.5 To delete a resource

- In the API Resource Paths for Product section of the product details window, locate the resource you want to disable, and then choose Delete in the Actions column.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

5.1.9.2.6 Multiple wildcards in API Product resource paths

When defining resource paths in API Product, you can include wildcards in multiple places in a resource path. For example, `/team/*/invoices/**` allows API calls with any one value after `/team` and any resource paths after `invoices/`. An allowed URI on an API call would be `proxyBasePath/team/finance/invoices/company/a`.

If after this release your existing API Product resource paths stop working as expected, set the following property on your organization to revert to the previous behavior: `features.enableStandardWildCardMatchForAPIProductResources = true`

5.1.9.3 JSON payloads in Assign Message and Raise Fault (Cloud 16.08.17)

When setting a JSON payload using an Assign Message or Raise Fault policy, users were sometimes required to use workarounds to ensure a JSON message was properly formatted at runtime, such as beginning the payload with a backslash "\" or specifying a variablePrefix and variableSuffix on the Payload element, even if no variables were used in the message.

With this enhancement, no workarounds are needed to ensure proper JSON message formatting, and variables can be specified using curly braces without creating invalid JSON. For example, the following inserts the value of message.content in the JSON message:

<Payload contentType="application/json">{"Message: " : "{message.content}"}</Payload>

If you used a workaround, your code will continue to work as is. You can also use variablePrefix and variableSuffix instead of curly braces to indicate variables.

5.1.10 Adding Developers to your Organization

Developers access your APIs through apps. When the developer registers an app, they receive a single API key that allows them to access all of the API products associated with the app. However, developers must be registered before they can register an app.

Developers typically have several ways of registering:

- By accessing a form that uses the SAP API Management API to register the developer. See Using the Management API to Publish APIs for more.
- By a back-end administrator using the SAP API Management UI, described below.

In the SAP API Management UI, select Publish > Developers to open the Developers page. You can see the apps they've created and the keys assigned to the developer. From this page, you can also manage the developer's information or delete them from your org.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Note

If you use the Developer Services portal, SAP API Management recommends that you log in to the portal as a portal administrator to manually create, edit, and delete developers on the portal, not in the SAP API Management UI. When you add a developer through the administrator interface on the portal, you can set the developer's password for the portal and trigger an automated email message sent to the developer. Also, any changes made to the developer's account on the portal are automatically sent to SAP API Management. When adding or modifying a developer through SAP API Management, no email is sent to the developer and you cannot set the password for the developer on the portal. Therefore, the developer must reset their password on the portal before they can log in to the portal.

5.1.10.1 Adding a Developer Manually Using the Management UI

Note

If you use the Developer Services portal, SAP API Management recommends that you create, edit, and delete developers on the portal itself, not in the SAP API Management UI. For more information,

1. Login to the SAP API Management.
2. Choose Publish > Developers in the menu.
3. Choose the (+) Developer button.
4. Enter the first name of your developer.
5. Enter the last name of your developer.
6. Enter the developer's email. This is the email address you use when sending keys and notifications to this developer.
7. Add any custom attributes for the developer. For each custom attribute:
8. Choose (+) Add Custom Attribute.
9. Enter the attribute name and value.
10. Select Save.
11. If you are using the Developer Services portal, synchronize the portal with the changes to SAP API Management as described below. When the new developer first logs in to the portal, they must use the password recovery process on the portal to set their portal password.

5.1.10.2 Editing a developer using the Management UI

Note

If you use the Developer Services portal, SAP API Management recommends that you create, edit, and delete developers on the portal itself, not in the SAP API Management UI.

1. Login to the SAP API Management UI.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

2. Choose Publish > Developers in the menu.
3. Choose the developer's entry in the Developers page.
4. Choose Edit.
5. Edit the developer's first and last name, as needed.
6. Add or remove any custom attributes.
7. Select Save.
8. If you are using the Developer Services portal, synchronize the portal with the changes to SAP API Management as described below.

5.1.10.3 Deleting a Developer Using the Management UI

> **Note**
>
> If you use the Developer Services portal, SAP API Management recommends that you create, edit, and delete developers on the portal itself, not in the SAP API Management UI. For more information, see Manage Users and Roles.

1. Login to the SAP API Management UI.
2. Choose Publish > Developers in the menu.
3. Locate the developer to delete in the list of developers.
4. If the number of Apps for the developer is zero, then the Delete button is enabled.
   - Select the Delete button to delete the user.
   - Confirm that you want to delete the developer.
5. If the number of Apps for the developer is not zero, then the Delete button is disabled. You must first delete all apps associated with the developer before you can delete the developer.
   - Choose the developer's entry in the Developers page.
   - Under Apps on the developer's page, select the app name.
   - Select Delete on the app's page, and confirm the deletion.
   - After deleting all the developer's apps, you can delete the developer.

If you are using the Developer Services portal, synchronize the portal with the changes to SAP Management UI as described below.

5.1.10.4 Synchronizing the portal with developers modified using the UI

If you are using a Developer Services portal to publish your APIs, changes made to developers through the SAP API Management UI are not pushed down to the portal. Therefore, you must log in to the portal as a portal administrator and synchronize the portal with SAP API Management for those changes to appear on the portal.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

> **Note**
>
> If you use the Developer Services portal, SAP API Management recommends that you create, edit, and delete developers on the portal itself, not in the SAP API Management UI. For more information, see Add and manage user accounts section in the Developer Portal Guide.

To synchronize the portal with a developer added Management Portal:

1. Log in to your portal as a user with admin or content creation privileges.
2. Select People in the Drupal administration menu.
3. Select the Dev Portal Developer Sync button at the top of the page to pull down any developers added on SAP API Management.

5.1.11 Registering Apps

This topic explains how to register and manage apps by using the SAP API Management UI.

5.1.11.1 Overview

Developers register apps to access your API products. When a developer registers an app, the developer selects the API products to associate with the app and SAP API Management generates an API key. Each app has a single API key that provides access to all API products associated with the app.

Apps allow you to control who can access your APIs. You can revoke an app's key, preventing it from accessing all API products. Or you can revoke access to a single API product associated with the app.

You can see all of your organization's apps in the SAP API Management UI on the Publish > Developer Apps summary page. This page displays performance data for each app, and general information on app keys. You can select a specific app from the table to get more detailed information, including the API products that app can access and the resources those products expose. You can also see the key associated with the app.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

5.1.11.2 Registering an App in the Management UI

Often, developers register their apps through your developer portal. However, if you do not use a portal or if you want to maintain complete control over the app registration process, you can register apps by using the SAP API Management UI. For example, you want to register apps for your internal development teams or on behalf of a developer without access to your portal. When you register an app, you decide which API products to associate with it. When the app is registered, SAP API Management automatically generates an API key to grant access to the selected API products. It is then up to you to determine how to pass that key to the app developer.

1. Any additional API products you associate with the app use the same key.
2. To register an app:
3. Login to the SAP API Management UI.
4. Choose the Publish tab, then Developer Apps.
5. Choose (+) Developer App on the Developer Apps page.
6. Enter an application name in the Display name field.
7. Select a developer from the Developer list.
   An app must be associated with a registered developer. If the developer does not appear in the list, you can register them. See Adding developers to your organization for more.
8. If necessary, enter the Callback URL.
   A callback URL is the location of a resource that belongs to the app. For example, if you are using OAuth, this could be the location of a login screen where users enter their username and password. This value s not required for all API products.
9. Optionally, add a descriptive note for the app.
10. Add any custom attributes for the app. For each custom attribute:
    - Choose (+) Add Custom Attribute.
    - Add the attribute name and value.
11. Choose Save.
    Your app is added to the apps list on the Developer Apps page.

5.1.11.3 Editing an App in the Management UI

To edit an app:

1. Login to the SAP API Management UI.
2. Choose the Publish tab, then Developer Apps.
3. Select an app in the Developer Apps summary table.
4. Choose Edit.
5. Edit the app's display name and callback URL, as needed.
6. Add, remove, or change a note, as needed.
7. Add, remove, or change any custom attributes, as needed.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Note

Be careful when you edit custom attributes. If you have a system that has dependencies on custom attributes, it might be impacted if you edit the custom attributes using the SAP API Management UI.

8. Choose Save.

5.1.11.4 Controlling Access to Products in the Management UI

You give an app access to your APIs by associating API products with the app. If you want to limit an app's access, you can temporarily revoke access to an API product, or permanently cut off access to a product by deleting the API product from the app.

Note

When you create an API product, you can also set its access mode to Internal only or Private. API products marked Internal only or Private do not appear to developers on the developer portal. To get access to these products, you manually add them to a developer's app from the SAP API Management UI.

5.1.11.4.1 Adding Access to an API product

You can expand the app's access to your APIs by associating the app with additional API products:

1. Login to the SAP API Management UI.
2. Choose the Publish tab, then Developer Apps.
3. Select an app in the Developer Apps summary table.
4. On the Developer App page, choose Edit.
5. In the Products section of the Edit Developer App page, choose + Product, select the product you want to add to the app, and choose the check mark icon.
6. Choose Save. The product is added to the app.

5.1.11.4.2 Revoking Access to an API product

In this procedure, you revoke access to this product and the resources it contains. You can re-enable access at any time.

1. Login to the SAP API Management UI.
2. Choose the Publish tab, then Developer Apps.
3. Select an app in the Developer Apps summary table.
4. On the Developer App page, choose Edit.
5. In the Products section of the Edit Developer App page, choose Revoke next to the product.
   The Revoke button changes to Approve in case you want to make the app part of that product again.
6. Choose Save.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

5.1.11.4.3 Deleting a Product from an App

In this procedure, you remove a product from an app.

1. Login to the SAP API Management UI.
2. Choose the Publish tab, then Developer Apps.
3. Select an app in the Developer Apps summary table.
4. On the Developer App page, choose Edit.
5. In the Products section of the Edit Developer App page, choose Remove next to the product.
6. Choose Save.

5.1.11.5 Manually Approve Keys

If you specified automatic key approval when you created the API product, a key is automatically approved for use by an app. If you specified manual key approval when creating the API product, you need to approve the key manually in the SAP API Management UI before the key can be used by the app to access the product.

In this procedure, you manually approve a product in an app.

1. Login to the SAP API Management UI.
2. Choose the Publish tab, then Developer Apps.
3. If any app is waiting for key approval, Key requested appears in the Key column of the page.
4. Select an app in the Developer Apps summary table with a pending key request.
5. On the Developer App page, choose Edit.
6. In the Products section of the Edit Developer App page, choose Approve next to the product.
7. Choose Save.

5.1.11.6 Regenerating a New Key

In some cases, you may need to regenerate a new consumer key for an app. For instance, you might do this if the security of the original keys is compromised. Note that if you simply remove products from an app, and then add the products back, the old consumer keys are reused. To regenerate new consumer keys:

1. Login to the SAP API Management UI.
2. Choose the Publish tab, then Developer Apps.
3. Select the app in the Developer Apps summary table for which you want to generate new consumer keys.
4. On the Developer App page, choose Regenerate Key.
5. In the pop-up, choose Regenerate Consumer Key.
6. Choose Save.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

5.1.12 Using the Management API to Publish APIs

This topic covers the definition of API products. API products enable developers to register apps that consume APIs using API keys and OAuth access tokens. API products are designed to enable you to 'bundle' API resources and then publish those bundles to different groups of developers. For example, you may need to publish one set of API resources to your partner developers, while you publish another bundle to external developers. API products enable you to perform this bundling on-the-fly, without requiring any changes to your APIs themselves. An additional benefit is that developer accesses can 'upgraded' and 'downgraded' without requiring developers to obtain new consumer keys for their apps.

> **Note**
>
> This topic demonstrates provisioning API products, apps, and developers using the SAP API Management API. You can also provision API products, developers, and apps using the SAP API Management UI.

5.1.12.1 Configuring API Products

You can configure API products using the SAP API Management UI or the SAP API Management API. The API enables you to create and manage API products programmatically.

To set up a minimal API product using the SAP API Management API, POST a payload to the /apiproducts resource in your organization.

The following request creates an API product called weather_free. The API product provides access to all APIs exposed by the API proxy called weatherapi that is deployed in the test environment. The approval type is set to auto--any request for access will be approved.

> **Caution**
>
> Remember to change the values for myname, mypass, and org_name to match the settings for your account.

```
$ curl -H "Content-Type:application/json" -X POST -d \
'{
  "approvalType": "auto",
  "displayName": "Free API Product",
  "name": "weather_free",
  "proxies": [ "weatherapi" ],
  "environments": [ "test" ]
}' \
http://<host:port>/v1/o/{org_name}/apiproducts \
-u myname:mypass
```

Sample response:

```
{
  "apiResources" : [ ],
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    "approvalType" : "auto",
    "attributes" : [ ],
    "createdAt" : 1362759663145,
    "createdBy" : "developer@yourcompany.com",
    "displayName" : "Free API Product",
    "environments" : [ "test" ],
    "lastModifiedAt" : 1362759663145,
    "lastModifiedBy" : "developer@yourcompany.com",
    "name" : "weather_free",
    "proxies" : [ "weatherapi" ],
    "scopes" : [ ]
}
```

You have now set up a very basic API product that can be used to restrict access to the API proxy named `weatherapi` running in the environment called `test`.

The API product created above implements the most basic scenario--it authorizes requests to an API proxy in an environment. It defines an API product that enables an authorized app to access any API resources exposed by the API proxy running in the test environment. API products expose additional configuration settings that enable you to customize access control to your APIs for different developer groups. For example you can create two API products that provide access to different API proxies. You can also create two API products that provide access to the same API proxies, but with different associated Quota settings.

5.1.12.2 API Product Configuration Settings

API products expose the following configuration options:

Name	Description	Default	Required?
apiResources	A comma-separated list of URIs 'bundled' into the API product. The URIs by default are mapped from the proxy.pathsuffix variable. The proxy path suffix is defined as the URI fragment following the ProxyEndpoint base path. For example, in the sample API product below, theapiResources element is defined to be/forecastrss. Since the BasePath defined for this API proxy is /weather, that means that only requests to/weather/forecastrss are permitted by this API product. You can define several API products to "bundle" API resources for different business or technical purposes. Wildcards (/** and /*) are supported. The double asterisk wildcard indicates that all sub-URIs are	N/A	No

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Default	Required?
	included. A single asterisk indicates that only URIs one level down are included.		
approvalType	Specifies how API keys are approved to access the APIs defined by the API product. If set tomanual, the key that is generated for app is in the 'pending' state. Such keys won't work until they have been explicitly approved. If set toauto, all keys are generated in the 'approved' and work right away.	N/A	Yes

attributes	Array of custom attributes that may be used to extend the default API product profile with customer-specific metadata. For example:
	"attributes": [
	{
	"name": "foo",
	"value": "foo"
	},
	{
	"name": "bar",
	"value": "bar"
	}
	]
		N/A	No

scopes	A comma-separated list of OAuth scopes that are validated at runtime. (SAP API Management validates that the scopes in any access token presented match the scope set in the API product.)	N/A	No
proxies	Named API proxies to which this API product is bound. By specifying proxies, you can bind the API resources listed in the API product to specific API proxies, preventing developers from accessing those URIs via other API proxies.	N/A	No. If not defined,apiResources must be explicitly defined, andflow.resource.namevariable set in AssignMessage policy.
environments	Named environments (for example 'test' or 'prod") to which this API product is bound. By specifying one or more environment, you can bind the API resources listed in the API product to specific environment, preventing developer from accessing those URIs via API proxies in another environment. This setting is used, for example, to prevent URIs protected by API proxies in 'prod' from being accessed by API proxies deployed in 'test'.	N/A	No. If not defined, api Resources must be explicitly defined, andflow.resource.namevariable set in

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Default	Required?
			AssignMessage policy.
quota	Number of requests permitted per app over the time interval specified.	N/A	No
quotaInterval	Number of time units over which quotas are evaluated	N/A	No
quotaTimeUnit	The time unit (minute, hour, day, or month) over which quotas are counted.	N/A	No
approvalType	Specifies how API keys are approved to access the APIs defined by the API product. If set tomanual, the key that is generated for app is in the 'pending' state. Such keys won't work until they have been explicitly approved. If set toauto, all keys are generated in the 'approved' and work right away.	N/A	Yes
quotaTimeUnit	The time unit (minute, hour, day, or month) over which quotas are counted.	N/A	Yes

```
$ curl -H "Content-Type:application/json" -X POST -d \
 '{
  "apiResources": [ "/forecastrss" ],
  "approvalType": "auto",
  "attributes":
    [ {"name": "myAttribute", "value": "myValue"} ],
  "description": "Free API Product",
  "displayName": "Free API Product",
  "name": "weather_free",
  "scopes": [],
  "proxies": [ "weatherapi" ],
  "environments": [ "test" ],
  "quota": "10",
  "quotaInterval": "2",
  "quotaTimeUnit": "hour" }' \
-u myname:mypass http://<host:port>/v1/o/{org_name}/apiproducts
```

Sample Response

```
{
  "apiResources" : [ "/forecastrss" ],
  "approvalType" : "auto",
  "attributes" : [ {
    "name" : "myAttribute",
    "value" : "myValue"
  },
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    "createdAt" : 1344454200828,
    "createdBy" : "admin@yourcompany.com",
    "description" : "Free API Product",
    "displayName" : "Free API Product",
    "lastModifiedAt" : 1344454200828,
    "lastModifiedBy" : "admin@yourcompany.com",
    "name" : "weather_free",
    "scopes" : [ ],
    "proxies": [ {'weatherapi'} ],
    "environments": [ {'test'} ],
    "quota": "10",
    "quotaInterval": "1",
    "quotaTimeUnit": "hour"}'
}
```

5.1.12.2.1 A note on scopes

A scope is a concept drawn from OAuth and maps roughly to the concept of a 'permission'. On SAP API Management, scopes are completely optional. You can use scopes to achieve finer-grained authorization. Every consumer key issued to an app is associated with a 'master scope'. The master scope is the set of all scopes in all API products for this the app has been approved. For apps approved to consume multiple API products, the master scope is the union of all scopes defined in the API products for which the consumer key has been approved.

5.1.12.3 Registering Developers

All apps belong to either developers or companies. Therefore, to create an app, you first need to register a developer or company.

> **Note**
>
> This section describes how to register apps for developers. You can use the Companies API to set up a company group and the Company Developers API to add developers to the company.

You can register a developer using the SAP API Management UI or using the SAP API Management API. Developers are registered in an organization by creating a profile. Note that the developer email that is included in the profile is used as the unique key for the developer throughout SAP API Management. Arbitrary attributes can be added to the developer profile as needed. The attributes are not interpreted by SAP API Management, but can be of use in custom analytics, custom policy enforcement, and so on.

For example, to register a profile for a developer whose email address is `ntesla@theremin.com`:

```
$ curl -H "Content-type:application/json" -X POST -d \
'{"email" : "ntesla@theremin.com",
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  "firstName" : "Nikola",
  "lastName" : "Tesla",
  "userName" : "theremin",
  "attributes" : [ { "name" : "project_type", "value" : "public"} ]
}' \
http://<host:port>/v1/o/{org_name}/developers \
-u myname:mypass
```

Sample Response

```
   {
         "email" : "ntesla@theremin.com",
         "firstName" : "Nikola",
         "lastName" : "Tesla",
         "userName" : "theremin",
         "organizationName" : "{org_name}",
         "status" : "active",
         "attributes" : [ {
           "name" : "project_type",
           "value" : "public"
         } ],
         "createdAt" : 1343189787717,
         "createdBy" : "admin@yourcompany.com",
         "lastModifiedAt" : 1343189787717,
         "lastModifiedBy" : "admin@yourcompany.com"
       }
```

5.1.12.4 Registering Developer Apps

Every app that is registered on SAP API Platform is associated with a developer and an API product. When an app is registered on behalf of a developer, SAP API Platform generates a "credential" (a consumer key and secret pair) that identifies the app. The app then must pass these credentials as part of every request to an API product associated with the app.

The following request registers an app for the developer you created above: ntesla@theremin.com. When registering an app you define a name for the app, a callbackUrl, and a list of one or more API products:

```
   $ curl -H "Content-type:application/json" -X POST -d \
   '{
     "apiProducts": [ "weather_free"],
     "callbackUrl" : "login.weatherapp.com",
     "name" : "weatherapp"}' \
   http://<host:port>/v1/o/{org_name}/developers/ntesla@theremin.com/apps \
   -u myname:mypass
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Note

The callbackUrl is used by some OAuth grant types (such as authorization code) to validate redirect requests from the app. If you use OAuth, then this value must be set to the same value as theredirect_uri used to make OAuth requests.

Sample Response

```
{
  "callbackUrl" : "login.weatherapp.com",
  "createdAt" : 1343190620890,
  "createdBy" : "admin@yourcompany.com",
  "credentials" : [ {
    "apiProducts" : [ {
      "apiproduct" : "weather_free",
      "status" : "approved"
    } ],
    "attributes" : [ ],
    "consumerKey" : "HQg0nCZ54adKobpqEJaE8FefGkdKFc2J",
    "consumerSecret" : "1eluIIdWG3JGDjE0",
    "status" : "approved"
  } ],
  "lastModifiedAt" : 1343190620890,
  "lastModifiedBy" : "admin@yourcompany.com",
  "name" : "weatherapp",
  "status" : "approved"
}
```

5.1.12.5 Managing Consumer Keys for Apps

5.1.12.5.1 Get the Consumer Key (the API Key) for the App

The credentials for an app, (API product, consumer key and secret) are returned as part of the app profile. An administrator of an organization can retrieve the consumer key at any time.

The app profile displays the value of the consumer key and secret, the status of the consumer key, as well as any API product associations for the key. As an admin, you can retrieve the consumer key profile at any time using the GET method:

```
$ curl -X GET -H "Accept: application/json" \
http://<host:port>/v1/o/{org_name}/developers/ntesla@theremin.com/apps/weatherapp/keys/HQg0nCZ54adKobpqEJaE8FefGkdKFc2J \
-u myname:mypass
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Sample Response

```
{
  "apiProducts" : [ {
    "apiproduct" : "weather_free",
    "status" : "approved"
  } ],
  "attributes" : [ ],
  "consumerKey" : "HQg0nCZ54adKobpqEJaE8FefGkdKFc2J",
  "consumerSecret" : "1eluIIdWG3JGDjE0",
  "status" : "approved"
}
```

5.1.12.5.2 Approving Consumer Keys

Setting approval type to manual enables you to control which developers can access the resources protected by API products. When API products have key approval set to manual, consumer keys must be explicitly approved. Keys can be explicitly approved using the following method call:

```
$ curl -X POST -H "Content-type:appilcation/octet-stream" \
http://<host:port>/v1/o/{org_name}/developers/ntesla@theremin.com/apps/weatherapp/keys/HQg0nCZ54adKobpqEJaE8FefGkdKFc2J?"action=approve" \
-u myname:mypass
```

Sample Response

```
{
  "apiProducts" : [ {
  "apiproduct" : "weather_free",
  "status" : "approved"
} ],
  "attributes" : [ ],
  "consumerKey" : "HQg0nCZ54adKobpqEJaE8FefGkdKFc2J",
  "consumerSecret" : "1eluIIdWG3JGDjE0",
  "status" : "approved"
}
```

5.1.12.5.3 Approving API Products for Consumer Keys

The association of an API product with a consumer key also has a status. For API access to be successful, the consumer key must be approved, and the consumer key must be approved for the appropriate API product. The association of a consumer key with an API product can be approved as follows:

```
$ curl -X POST -H "Content-type:application/octet-stream" \
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
http://<host:port>/v1/o/{org_name}/developers/ntesla@theremin.com/apps/weatherapp/keys/HQg0nCZ54adKobpqEJaE8FefGkdKFc2J/apiproducts/weather_free?"action=approve" \
-u myname:mypass
```

This cURL command does not return a response.

5.1.12.6 Enforcing API Product Settings

> **Note**
>
> API product settings are enforced by matching policies, such as VerifyAPIKey, OAuthV2, and Quota. Until those policies are attached to API proxies, API products and quotas will not be enforced by SAP API Management.

For API products to be enforced, one of the following policy types must be attached to the API proxy flow:

- VerifyAPIKey: Takes a reference to an API key, verifies that it represents a valid app, and matches the API product. See Enforce access control using VerifyAPIKey for more.
- OAuthV1, “VerifyAccessToken” operation: Verifies the signature, validates an OAuth 1.0a access token and “consumer key”, and matches the app to the API product. See Authorize requests using OAuth 1.0a for more.
- OAuthV2, “VerifyAccessToken” operation: Verifies that the OAuth 2.0 access token is valid, matches the token to the app, verifies that the app is valid, and then matches the app to an API product. See Authorize requests using OAuth 2.0 for more.
- GetAPIProduct: Enables policies to use API product metadata without necessarily verifying a token.

Once policies and API products are configured, the following process is executed by SAP API Management:

1. A request is received by SAP API Management and routed to the appropriate API proxy.
2. A policy is executed that verifies the API key or OAuth access token presented by the client.
3. SAP API Management resolves the API Key or access token to an app profile.
4. SAP API Management resolves the list (if any) of API products associated with the app.
5. The first API product that matches is used to populate Quota variables.
6. If no API product matches the API key or access token, then the request is rejected.
7. SAP API Management enforces URI-based access control (environment, API proxy, and URI path) based on the API product settings, along with Quota settings.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6 Analyzing APIs

6.1 Analytics Services Overview

This topic introduces SAP API Management Analytics Services.

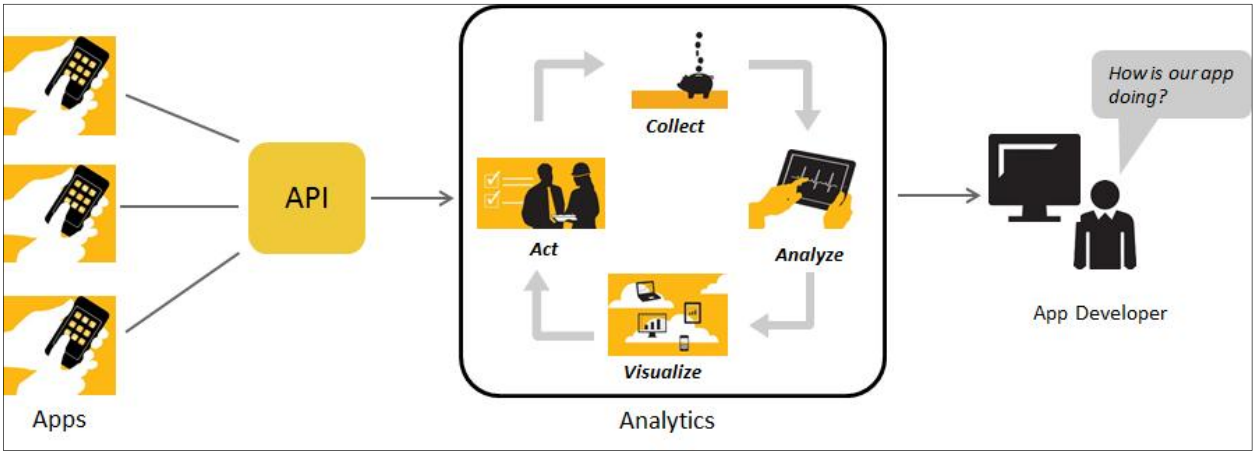
6.1.1 Improve your API through Analytics

SAP API Management Analytics Services collects and analyzes a wealth of information that flows through APIs. This information is gathered, analyzed, and provided to you immediately, in real time. How is your API traffic trending over time? Who are your top developers? When is API response time fastest? Slowest? Are you attracting more developers?

The answers to questions like these help you improve your APIs, attract the right app developers, troubleshoot problems, and, ultimately, make better business decisions related to your API program.

6.1.2 Insight that leads to action

Let's say your API has gained wide adoption. It's popular. You have attracted a number of talented, creative app developers and people are downloading and installing their apps. Obviously, the API team is very interested in how the API is performing, how it's being used, and how to plan for improvements.



To plan those improvements, the API team needs to know as much as they can about how the API is used in the real world. How is it performing? Who is using it? What are the traffic patterns? They need insight. They need to know:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

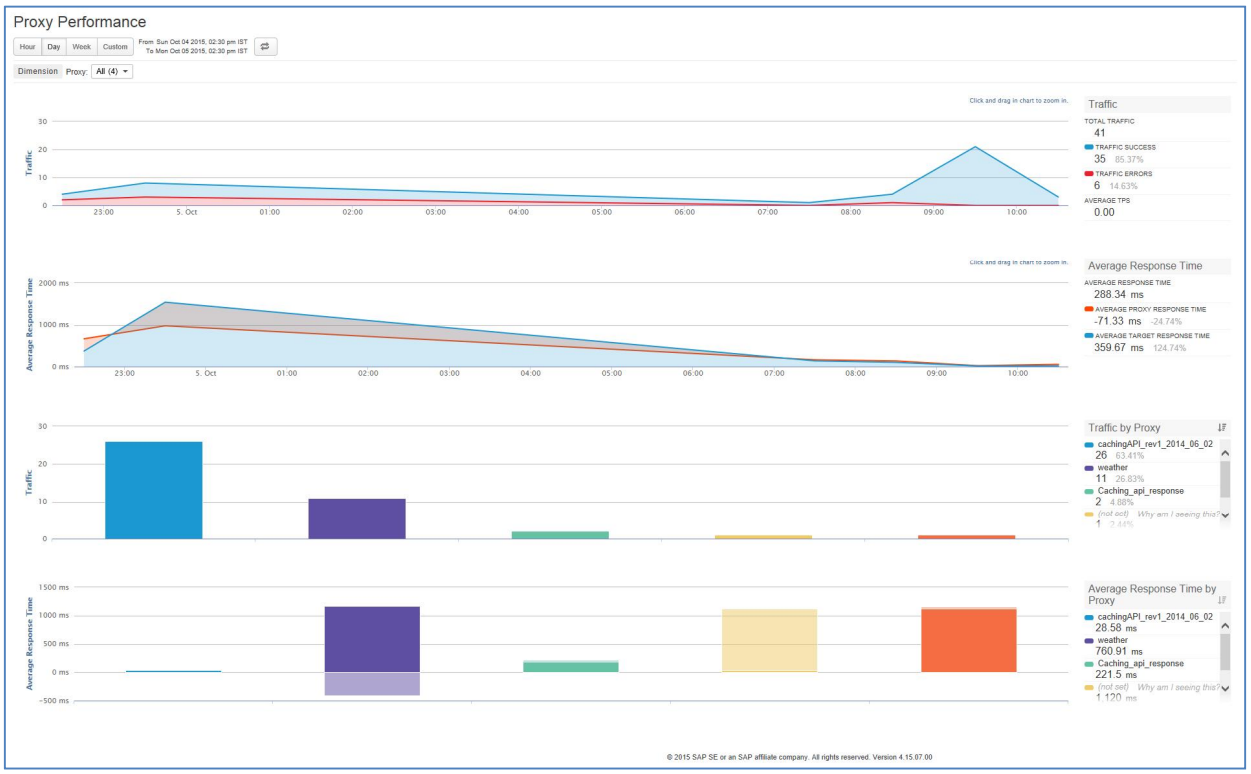
- What's changed? Knowing when and where something about an API changed helps you to take corrective action.
- What are our top apps? You need to know who/which apps are using your API.
- Who are our best app developers? Your best users and developers might be operating with you across multiple apps and devices, such as website, mobile apps, video game consoles, and tablets.
- Which API methods are most popular? Your API is a product; you want to understand its usage to help pivot your business and make future investment decisions.
- How much API capacity will be needed next year? Capacity planning is difficult. What if your API goes viral and you have millions of users? You need trending usage data to help make these decisions. In this context, you want to see indicators in the trend charts like the number of requests (the load hitting the backend systems), volume of data (especially for API responses), and so on.
- Why is the API down? There's probably not one thing to tell you why an API is down, but the shape of your graphs can help. Do your charts show catastrophic failure or a slow lead-in to failure?

Through a continual process of collecting, analyzing, and visualizing API metrics, Analytics Services helps your API team act to improve their APIs. Out-of-the-box, Analytics Services supports and encourages this pattern of collect, analyze, visualize, and act.

6.1.3 What kind of data is collected and analyzed?

Analytics Services collects and analyzes a broad spectrum of data that flows across APIs. For example, while your API is deployed, the analytics system is constantly counting and analyzing metrics like the number of messages received, number of developers, response times, errors, number of apps in use, request size, and many others. For a good introduction to metrics and how you can use them, see Use the analytics API to measure API program performance. For a complete listing of metrics, see Analytics Dimensions and Metrics.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.



You can also create custom metrics by wiring together features of Analytics Services and API Platform. For example, you can create a policy designed to extract certain information from an API (from request headers, for instance). Once extracted, you can use the analytics API to analyze and return the data. For example, you might ask: When does the highest response time usually occur for a specific API? The answer to a question like this might help your Ops team plan for anticipated network loads. See Analyze API message content using custom analytics for a nice end-to-end example demonstrating this technique.

6.1.4 A new breed of data

You've stored and collected metrics on system performance, you've collected error logs, you've collected user data, and you've collected many other kinds of data related to your business systems and the applications that run on them. Often this data is collected and stored in data warehouses where it is organized and possibly analyzed in predictable ways according to traditional models like ETL (extract, transform, load).

With the rapid adoption of APIs and a mobile apps economy, understanding what to measure is less predictable than it once was. Third-party app developers have access to your APIs. Users are on the go. They have smart phones, tablets, and other mobile devices. Some may never use a browser. In this rapidly shifting environment, you need to know exactly how, when, and where they are interacting with your APIs. You need to know how your APIs are performing right now, in real time.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.1.5 Put Analytics up front (a best practice)

SAP API Management encourages API teams to put analytics up front. Think of analytics as your API's peer, as an integral part of your API program from the beginning, not as something you consider or decide to add later in your API program.

Analytics can provide key information at the beginning of your API program to help you improve your API and the experience of app developers who use it. As your API program matures, analytics lends insight that can inform key business decisions.

6.1.6 Your API at a glance

Analytics Services provides several visualization tools, including the Dashboard, custom reports and tools that visualize trends in API proxy performance. For now, let's take a quick look at three of these tools: the Dashboard, proxy resource performance view, and custom reports.

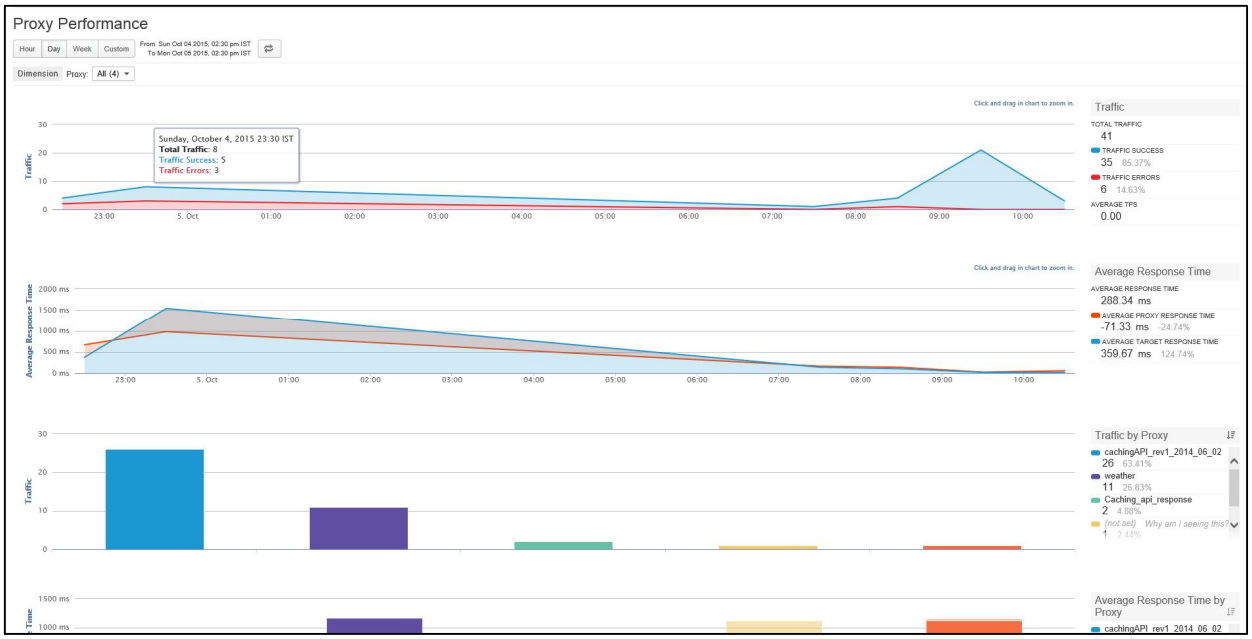
6.1.6.1 The Dashboard

The analytics Dashboard gives an overall view of your entire API program. You can toggle the dashboard between the API program view, which gives you a sense of overall API performance, and API top performer view, which tells you about your top performing APIs, apps, developers, and API products. You can select a time interval (hour, day, week, month) over which the data is plotted or specify a custom timespan. You can also mouse over any point on a plot to see in greater detail what was happening at that point.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.1.6.2 Proxy resource performance view

You also can track performance metrics for individual URIs (resources) for a specific API proxy. This view lets you plot trends in traffic, response time, and other metrics for each individual resource.



6.1.6.3 Custom reports

The custom report feature lets you select, combine, filter, and drill down into specific API metrics. You can generate reports that contain the exact data that you want to see. Reports can be viewed in the UI, exported to CSV files. You can schedule reports to be generated daily and emailed to you or to others. In addition, you can create your own analytics dashboard and populate it with custom reports that you select.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.1.7 Fetch analytics data with the REST API

Using the Analytics Services API, you can access a wide variety of operational and business data that flows across your APIs. For example, using REST calls, you can determine which APIs are performing well or poorly, which developers are delivering the highest value traffic, and which apps are causing the most issues for your backend services.

> **Info**
>
> The analytics dashboard is implemented using the REST API.

You can use the REST API when you need to automate certain analytics functions, such as retrieving metrics periodically using an automation client or script. You can also use the API to build your own visualizations in the form of custom widgets that you can embed in portals or custom apps. The API can also be used to provide custom analytics to certain app developers or communities of app developers.

Here's an example analytics REST API call:

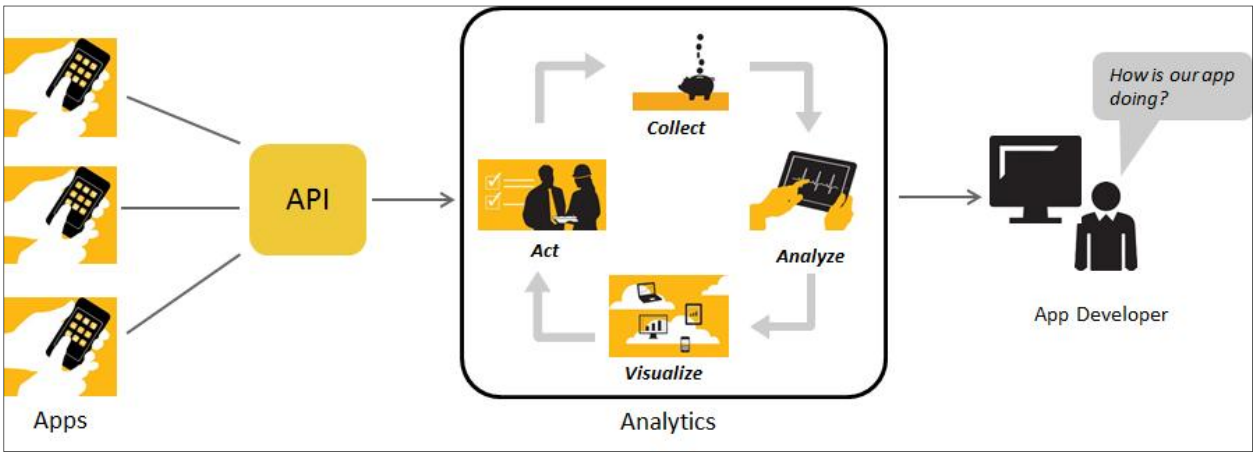
```
$ curl
http://<host:port>/v1/o/{org_name}/environments/test/stats/apis?"select=sum(message
_count)&timeRange=8/24/2013%2000:00~9/25/2013%2000:00&timeUnit=day" \
-u myname:mypass
```

This call tells you how many request messages were received on a daily basis by all API proxies between 8/24/20xx and 9/25/20xx. The response provides a snapshot of the request message throughput to all APIs for traffic received between 8/24/20xx and 9/25/20xx

To read all about the REST API for analytics, see Use the analytics API to measure API program performance.

6.1.8 Share data with app developers

When you think of app developers as your customers, you'll want to make sure they have the tools and information available to them to make the best use of your API. Ultimately, you want the people who use their apps (and your products and services) to be happy.



CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Analytics Services provides a wealth of information that app developers can use to improve their apps. App developers are not only concerned about the quality of their apps; they are very interested in your API. How is it performing? Is it error prone? Is it sometimes unavailable? How does it respond to traffic spikes? Answers to these questions are crucial to the app developer who wants to deliver the best possible experience to her (your) customers!

First, the developer portal includes an optional App Performance page designed to give app developers access to important metrics for their apps. This page gives developers information about:

- Errors: Which API errors is my app seeing? What is the rate of errors? Where are they appearing in the code?
- Performance: Is the API slow right now? Which API methods are typically slow or slow at the moment?
- Availability: Is the API up or down right now? When will the API be back up? Why was the API down?
- Quota: Does your API have a quota? And as the app developer, how am I doing against the quota? Am I violating the quota?

You can also create custom reports and share them with your app developers. For more information, see Build custom reports.

Finally, the analytics REST API can be made available to selected app developers. This API lets them access analytics data for a wide array of metrics. See Use the analytics API to measure API program performance.

6.1.8.1 Receive daily summary reports

All Organization Administrators are automatically subscribed to receive daily analytics summary reports through email.

Users in the Organization Administrator role can opt in and out of receiving daily analytics reports in the management UI. Select username > User Settings, and select or deselect the Receive daily analytics summary report option.

Organization Administrators can also use the /stats/preferences/dailysummaryreport API to unsubscribe (opt out) from receiving daily analytics reports, or subscribe (opt in).

For example, to opt out, use:

```
https://<host:port>/v1/organizations/{org}/stats/preferences/reports/dailysummaryreport?optin=false
```

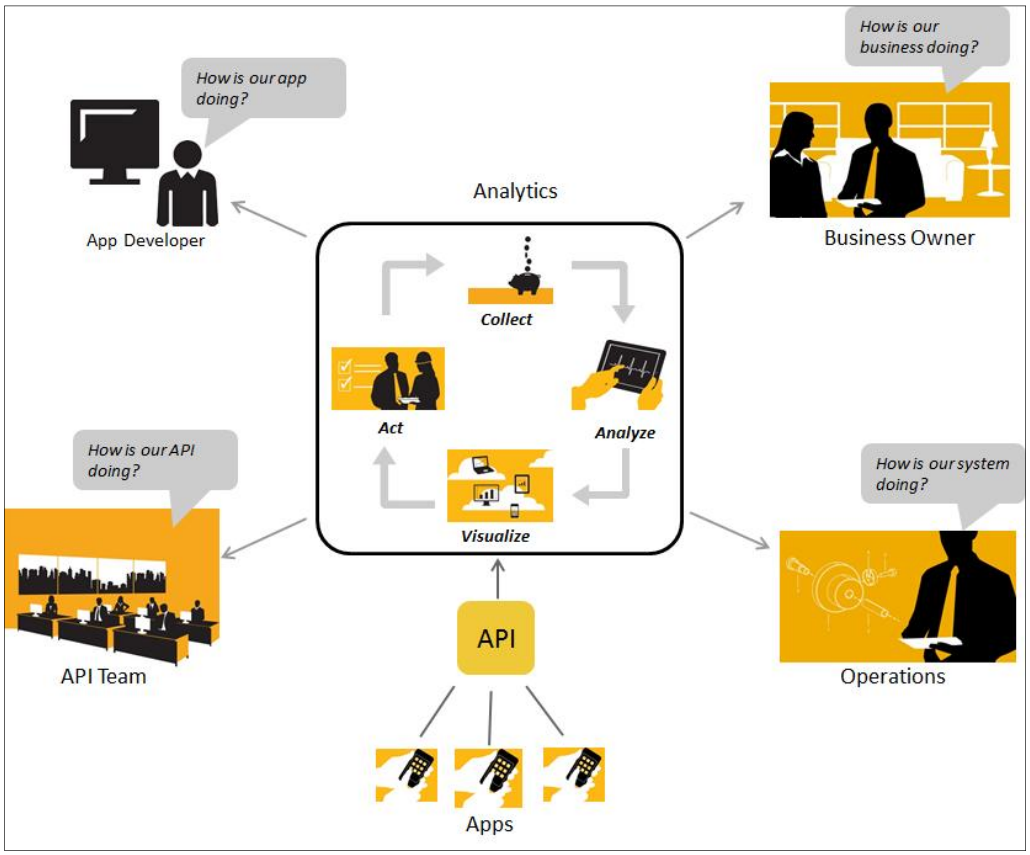
6.1.8.2 Export/import custom report definitions

You can use Edge APIs to export and import custom reports from one org environment to another. This useful technique lets you reuse custom report designs that you like in different orgs and environments. Furthermore, you can store the design (a simple JSON text file) in your CSV.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.1.8.3 API Analytics Helps Everyone Improve

From the app developer to the business owner, API analytics help everyone improve.



- API Team - The API team is tapping into internal systems to create interesting APIs. The API team wants to know how the API program is doing overall, how individual APIs are doing, and how to improve their APIs.

- App Developers - By sharing analytics information with app developers, you get better apps. These developers are innovating with your API and building creative apps that help drive revenue to your enterprise. Analytics help app developers know how their apps are doing and how much they are contributing to the bottom line of your enterprise. App developers want to know how they can improve their apps. Ultimately, everyone wants happy end users.

- Ops Team - Equally, the operations team wants to understand traffic patterns and anticipate when to add backend resources or make other critical adjustments.

- Business Owner - The business owner wants to see how her API investment is paying off and where to invest API dollars in the future. Ultimately, the business owner wants to improve her business.

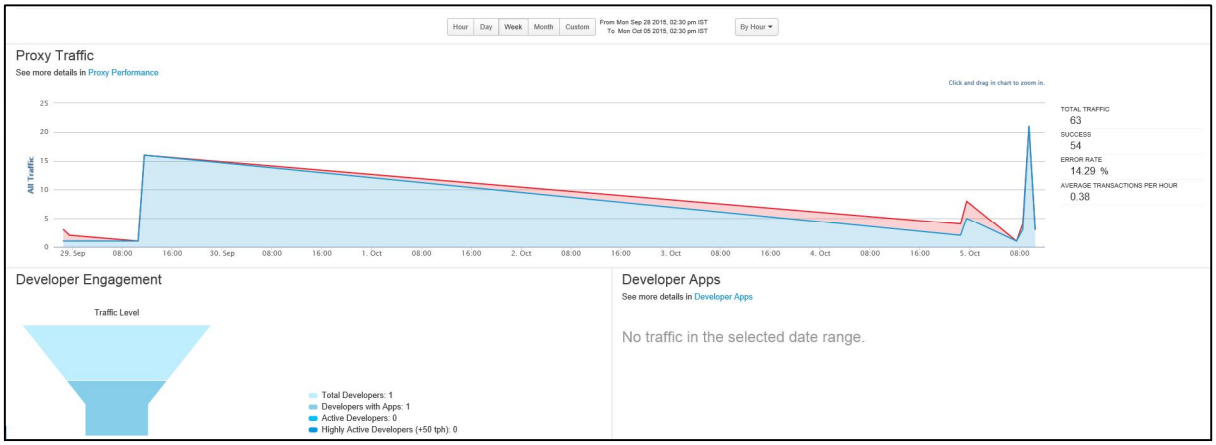
CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.2 Dashboard Homepage

6.2.1 SAP API Management Dashboard

The SAP API Management dashboard includes three views:

- Proxy Traffic
- Developer Engagement
- Developer Apps



6.2.2 API Dashboards

These dashboards highlight invaluable information about your API's health and performance.

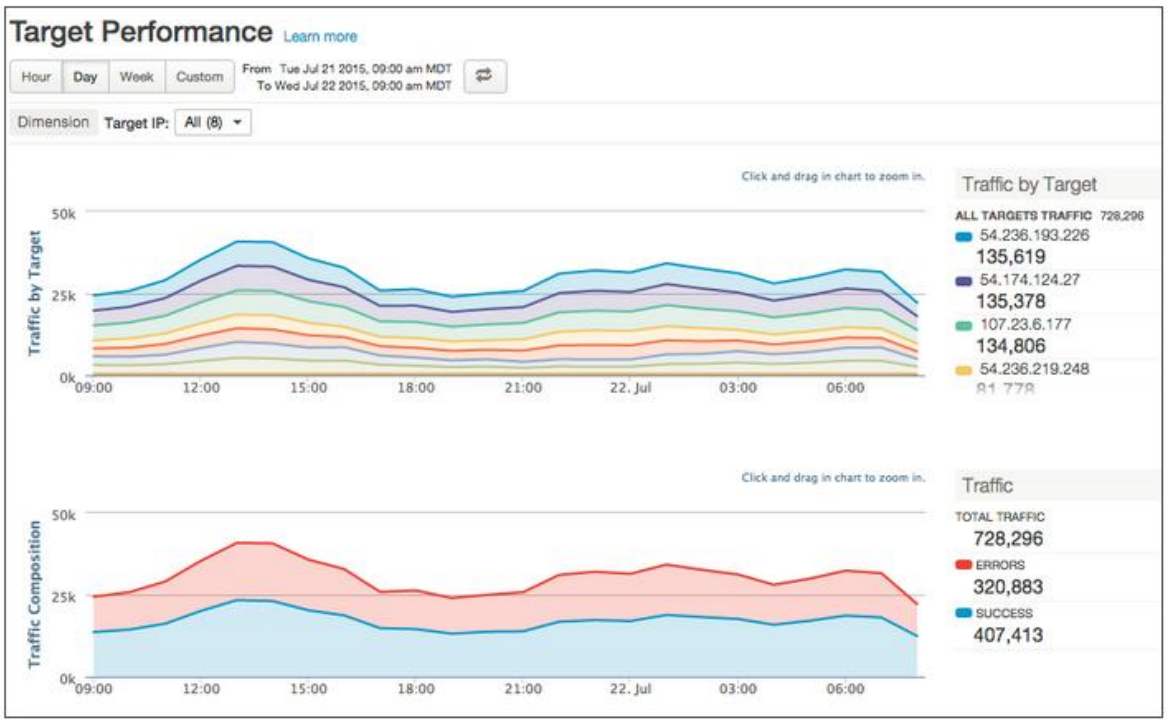
- API Proxy Performance

  API Proxy Performance tracks metrics like latency, API proxy errors, transactions per second and cache errors.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- Target Performance

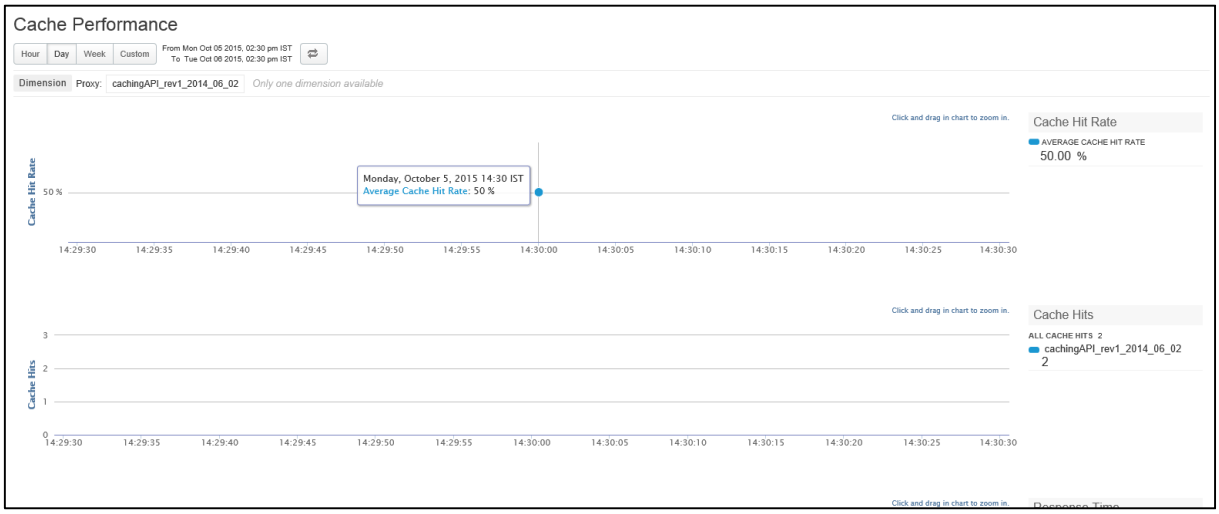
  Target Performance dashboard helps you visualize traffic patterns and performance metrics for API proxy backend targets.



© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- Cache Performance

  Cache Performance shows you metrics like cache hit rate, cache hits by API, cache hits by apps, total response time and others.

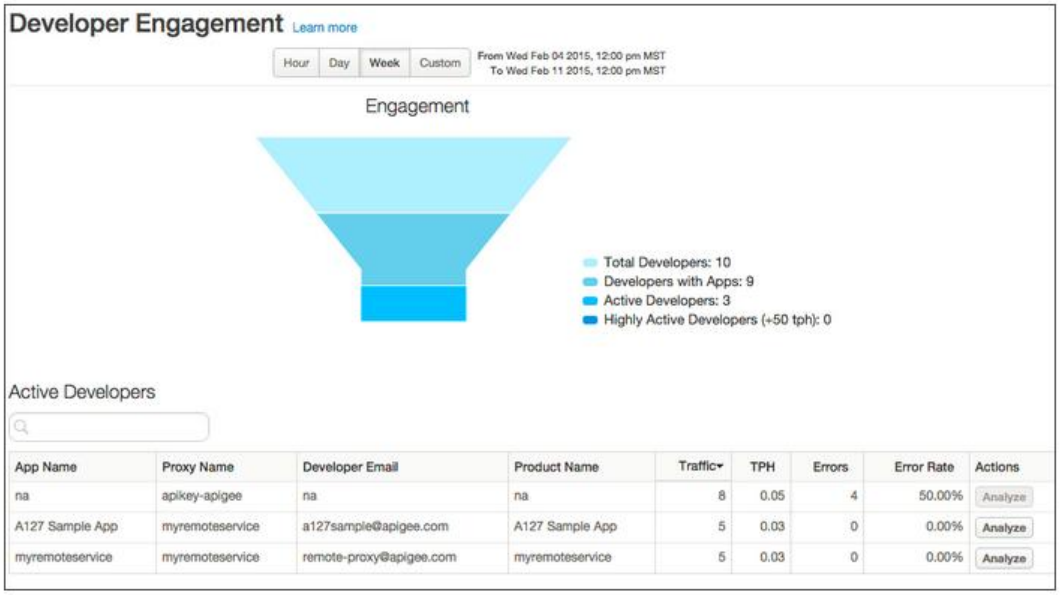


6.2.3 Developer Dashboard

These dashboards help you understand your API traffic patterns and adoption from multiple perspectives.

- Developer Engagement

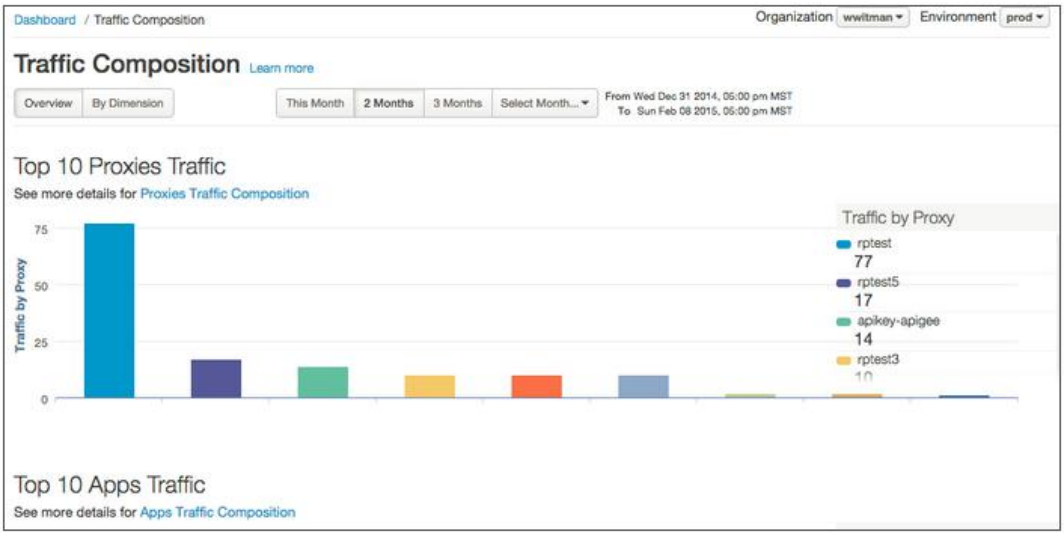
  The Developer Engagement dashboard tells you which of your registered app developers are generating the most API traffic.



© 2014 SAP SE or an SAP affiliate company. All rights reserved.

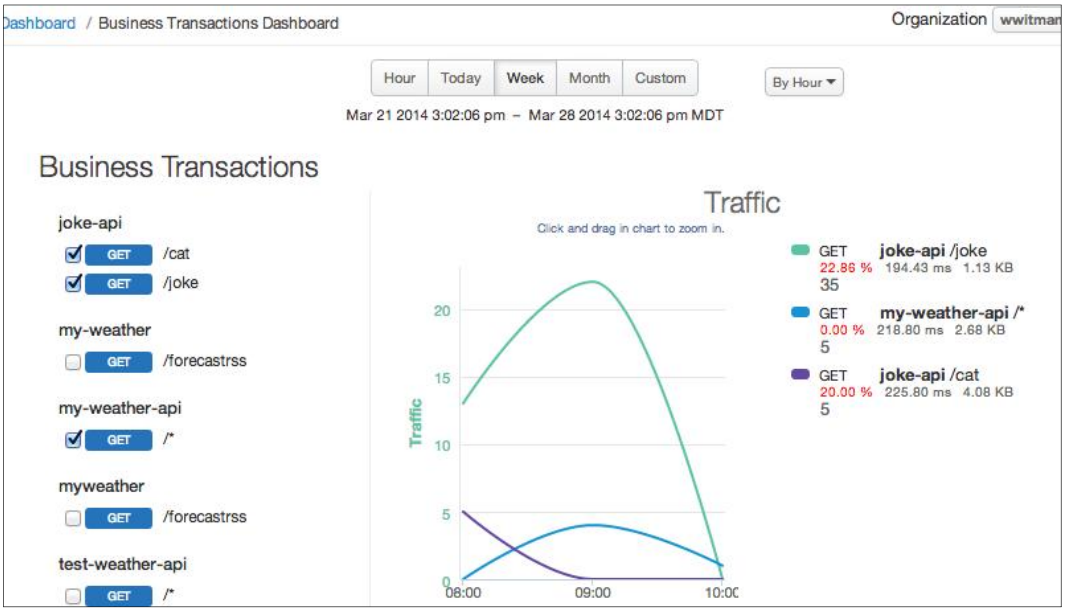
- Traffic Composition

  Traffic composition plots contribution over time of your most valuable apps, developers, APIs and products to your API program.



- Business Transactions

  Business Transactions tracks specific patterns in your API traffic and shows correlations between patterns from different API proxies.



CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.2.4 Exploration Dashboards

The Custom Reports tool and related dashboards let you generate and view reports based on metrics and dimensions you choose.

- Create Custom Reports

  Custom reports allows you to generate plots and charts from a broad range of API metrics including total response time, traffic flows, errors, cache hits and many others.

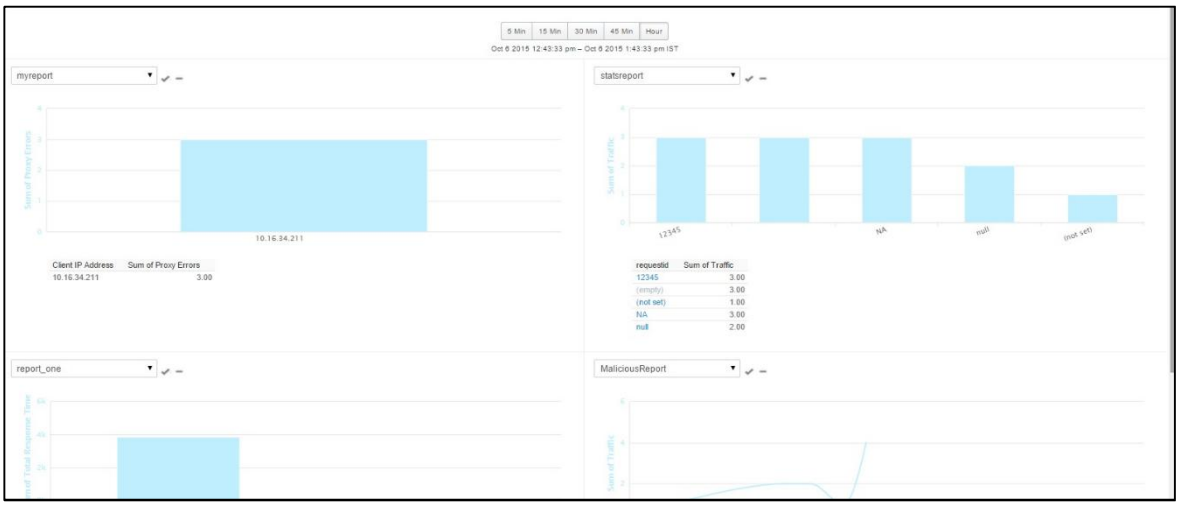
Report Name	Environment	Metric	Report Description	Last Modified	Actions
TPS	prod	Sum of Policy Errors, Sum of Cache Hit, Average of Total Response Time	cool	May 19, 2014 2:44:01 AM	Delete Roles
avgresponse	prod	Average of Total Response Time	test	May 9, 2014 4:59:33 PM	Delete Roles
Apps Traffic	prod	Sum of Traffic		May 7, 2014 1:00:54 PM	Delete Roles
WB: Daily Call Volume	prod	Sum of Traffic	test	May 7, 2014 1:00:54 PM	Delete Roles
TPS Report	prod	Sum of Average Transactions per Second		May 7, 2014 1:00:28 PM	Delete Roles
organizationtest	prod	Sum of Traffic	test	Mar 25, 2014 6:30:22 AM	Delete Roles
Traffic		Sum of Traffic, Sum of Proxy Errors	To track the API calls made for APIM	Mar 25, 2014 5:13:21 AM	Delete Roles
docstest	prod	Sum of Average Transactions per Second	docstest	Mar 19, 2014 2:22:46 PM	Delete Roles
API-Model	prod	Sum of Traffic	To track the API calls made for APIM	Mar 18, 2014 5:52:58 PM	Delete Roles
r25	prod	Sum of Traffic	test	Feb 28, 2014 3:25:48 AM	Delete Roles
test report	prod	Sum of Traffic		Jan 23, 2014 5:03:58 PM	Delete Roles

- Reports Dashboard

  Reports Dashboard allows you to see the results of the customized reports

- Reports

  Reports allows you to place up to four custom report dashboards in one page.



© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.3 Analytics Dashboard

6.3.1 Using the analytics dashboards

Analytics dashboards help you see and detect changes in your API ecosystem at a glance. The ability to see what has changed over time helps you identify problems and take corrective action quickly.

For a quick overview of Analytics Services, who uses them, and why, see Analytics Services overview.

6.3.1.1.1 What you'll learn in this topic

This topic explains how some of the common features you'll find in all the dashboards. After reading this topic, you will understand:

- The general layout of all dashboards
- Common features all dashboards share and how to use them
- A few tricks and tips that will help you get the most out of the analytics dashboards

6.3.1.1.2 What can you learn from the dashboards?

Has there been a sudden spike or drop off in API traffic? Which app developers are most successful? What is the adoption rate of your API among developers? Which API methods are most popular? The SAP API Platform Analytics dashboards are designed specifically to answer questions like these.

In the background, SAP API Management collects information as data passes through your APIs. The dashboards provide a powerful way to use this data immediately. If you see something of interest in a graph or chart, an anomaly or sudden change, you can then drill deeper to uncover as much detail as you require. If you notice that a particular developer is experiencing a lot of errors or a sudden drop in traffic, you can contact that developer proactively. Dashboards give you insight into your APIs that allows you to take action.

6.3.1.1.3 Can I customize the dashboards?

Yes, many dashboards let you select which metrics to analyze, date ranges, data aggregation intervals, and many other variables. If the built-in dashboards do not suit your needs, you can create custom reports, which are dashboards you create by selecting the analytic dimensions and metrics that you wish to analyze. Custom reports let you "drill down" into your API's analytic data until you achieve the granularity you require.

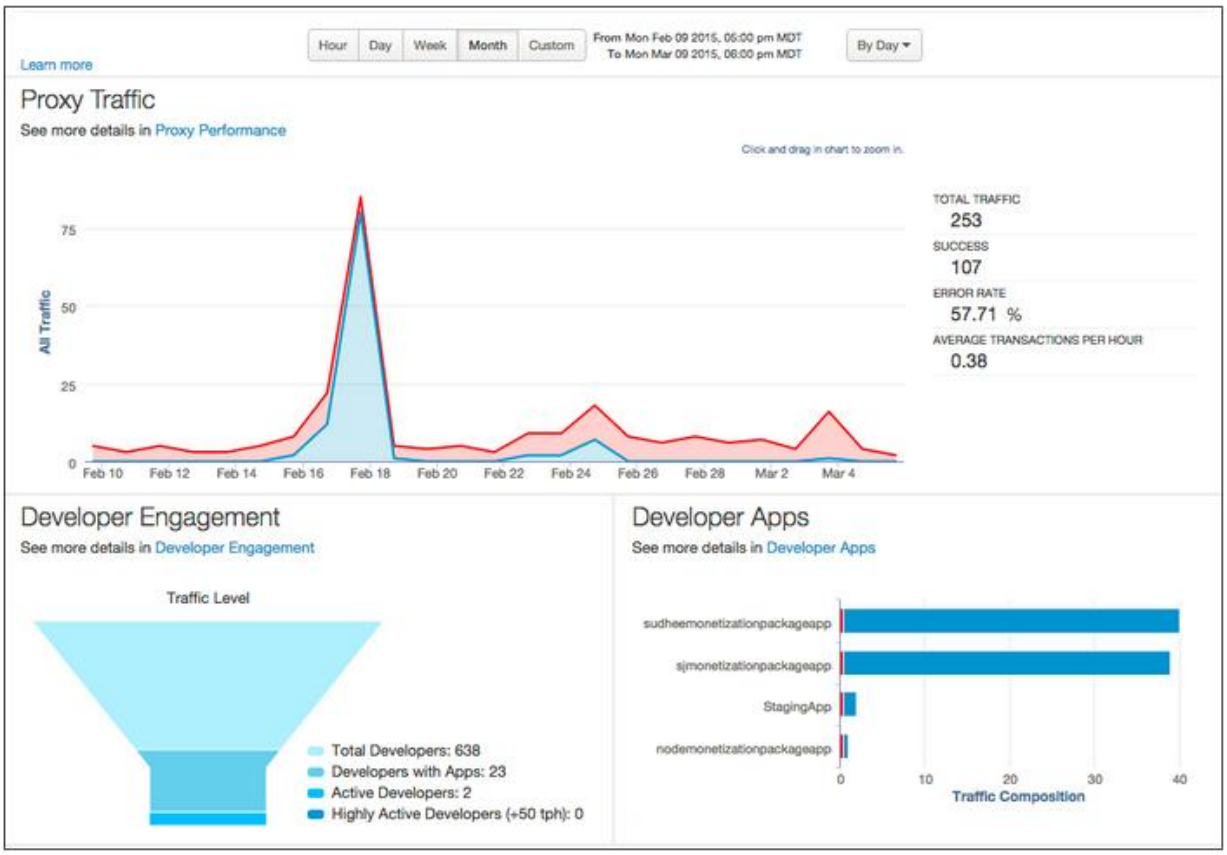
CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.3.1.1.4 How are the dashboards organized?

Most of the Analytics dashboards are organized and function much like the one below. If you understand how to use this dashboard, you'll be comfortable navigating the others.

- Data range and aggregation settings - Let you select the of dates for which to display data and the aggregation or "granularity" of data to display.
- Graph syles - All dashboards include one or more graphical charts, including line graphs, bar graphs, and pie charts.

> **Note**
>
> When you select a smaller aggregation interval the larger the dataset the dashboard is working with. Performance tends to increase when you select a larger aggregation interval. For example, performance is greater if you select By Day rather than By Minute.



6.3.1.1.5 What are the most common features in the dashboards?

Dashboards have a set of common features, including time range and data aggregation settings, view togbgles, choose and drag zooming on charts, mouse-over hover for more details on charts and other regions, and

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

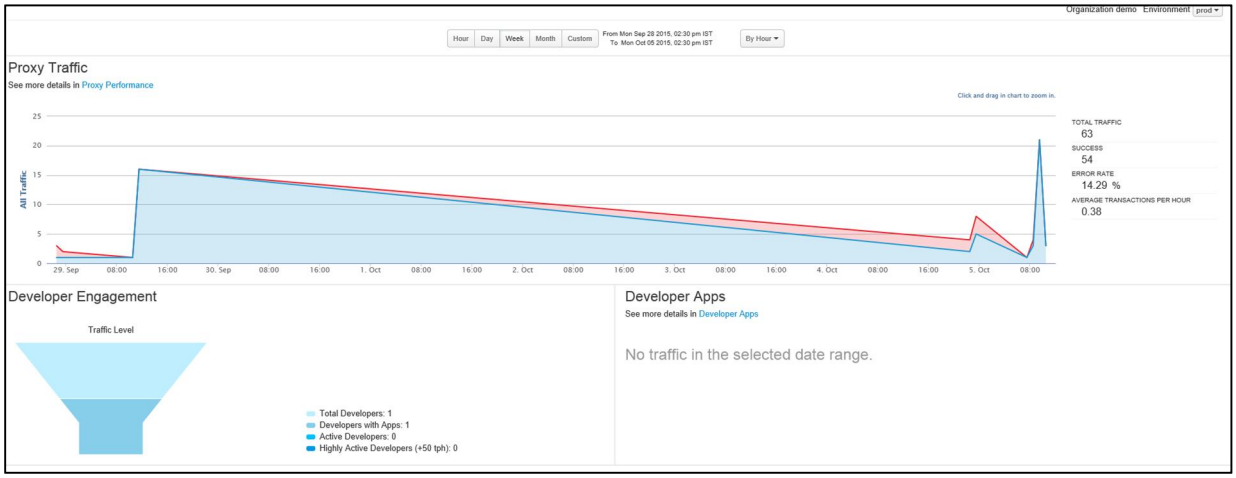
selectors for choosing the data to display in a chart. If you understand how to use one kind of dashboard, you'll be comfortable using the others.

- Toggle the view - Some dashboards let you toggle between distinct views.
- Set the time range - You can select the time range over which the dashboard displays its data. Select Custom to use a calendar-style date picker to select the interval.
- Select the data aggregation interval - The aggregation interval ranges from minute, hour, day, week, and month.

  > **Note**
  >
  > When you select a smaller aggregation interval the larger the dataset the dashboard is working with. Performance tends to increase when you select a larger aggregation interval. For example, performance is greater if you select By Day rather than By Minute.

- Zoom in - You can zoom in on chart data by choosing and dragging.
- Hover mouse over graphs - You can mouse over any point on a graph for more context about the data at that point.
- Hover mouse over labels - You can sometimes hover over a label for more details. Look for the "?" cursor as you mouse around.
- Export data to file - Some dashboards let you export CSV data format to a file. Look for the Export drop down menu or the export icon:

The following figure highlights these feature areas:



Here's another dashboard that includes additional features that you'll see in some other dashboards:

- Refresh button - Updates the dashboard with the latest data.
- Export data to a file - Select the data format from the Export menu.
- Feature selections - Lets you show or hide certain features of the dashboard like investigating anomolies and showing moving averages.
- Dropdown menu for metric selection - Let's you pick the data dimension that you wish to plot.
- Plot selectors - Lets you select which set of data you wish to plot. In the example below, you can select which API proxy you wish to plot.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

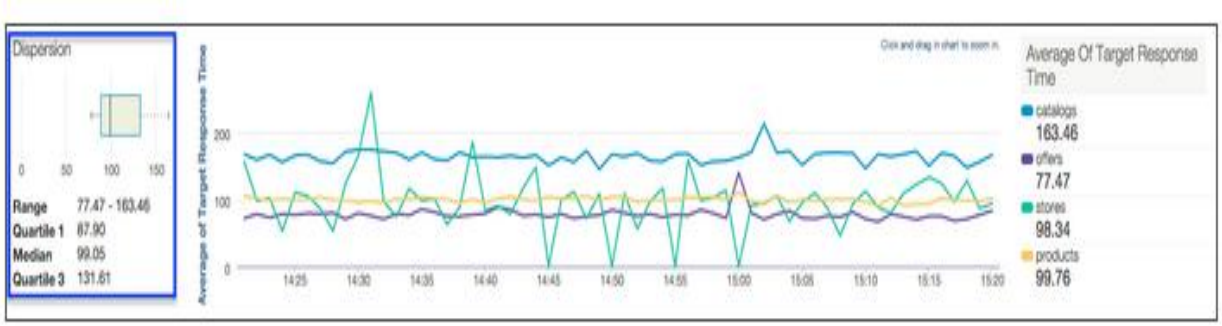
6.3.1.1.6 Viewing moving averages and alerts

In addition to the Investigate Anomalies checkbox, there are two checkboxes displayed for the APIs on the API Proxies page or for an API on its detail page.

Show Moving Averages	Show Alerts
Check this checkbox to view a moving average for the API. You can check this checkbox for multiple APIs to view a moving average that includes the set of these APIs. A moving average is a series of averages taken over sucessive subsets of a complete set of data. It's especially useful in viewing trends. The moving average is displayed as a band whose limits are +-20% of the calculated moving average data points.	Select this checkbox (on the API Proxies page) to view the number of times that the moving average for the API exceeded the +-20% limit.

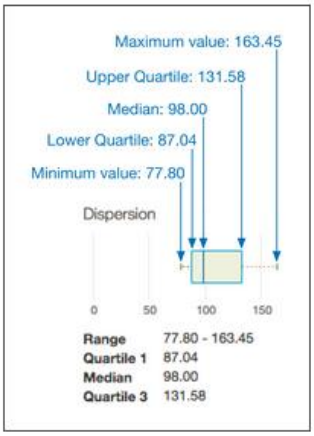
6.3.1.1.7 Reading dispersion box plots

When an analytics report illustrates an average, as well as minimum and maximum values, we display an accompanying dispersion box plot, as called out in the custom report below.



At a glance, a dispersion box plot enables you to see the central tendency and dispersion of your data. A dispersion box plot surfaces the five key numbers when it comes to illustrating averaged analytics data:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.



In this example, the area within the box indicates the most typical average target response times experienced by your traffic — 50% of your traffic, to be exact.

The line extending from the left side of the box indicates the average target response times experienced by 25% of your traffic.

The line extending from the right of the box indicates the average target response times experienced by the remaining 25% of your traffic.

The longer these lines, or “whiskers,” the more extreme your outlier values.

6.3.2 SAP API Management Dashboard

6.3.2.1 What is the SAP API Management dashboard?

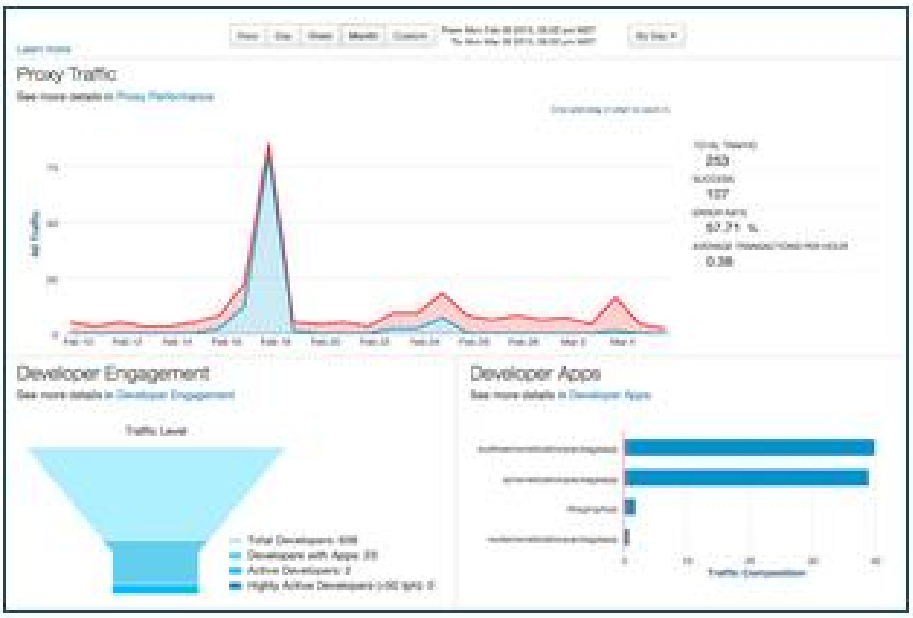
The SAP API Management dashboard is the first thing you see when you log in to SAP API Management. It gives you end-to-end visibility to monitor, measure, and manage the success of your API program.

The SAP API Management dashboard includes three views:

- Proxy Traffic - Monitors how your APIs are growing, the engagement of your developers, and the rates of developer adoption.
- Developer Engagement - Gives a quick sense of how developers are engaged with your APIs.
- Developer Apps - Breaks down API traffic by developer app. Shows you which apps are receiving the most traffic.

To go to this dashboard at any time, select Dashboard from the main SAP API Management menu.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.



6.3.2.1.1 What does the API Program view tell me?

Metric	Description
API traffic	All the API traffic for an organization over the selected period of time. Traffic, also called "throughput", represents the number of API requests and responses seen by an API organization.
Developer Engagement	Measures the average number of API calls for registered developers during the selected period of time. The ratio among traffic levels for high, medium, low, and inactive developers using your APIs for the selected time period. Also shown are the total number of developers who have signed up to use your APIs, and the percentage change of developer signups between this week and last week. Note: It's possible that the Developer Engagement could reflect higher numbers for a Week timespan than for a Month. This is because engagement is calculated based on the average number of calls during a specific period of time (week/month). You could have a large number of developers register their apps, but for a period of time the apps are not

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

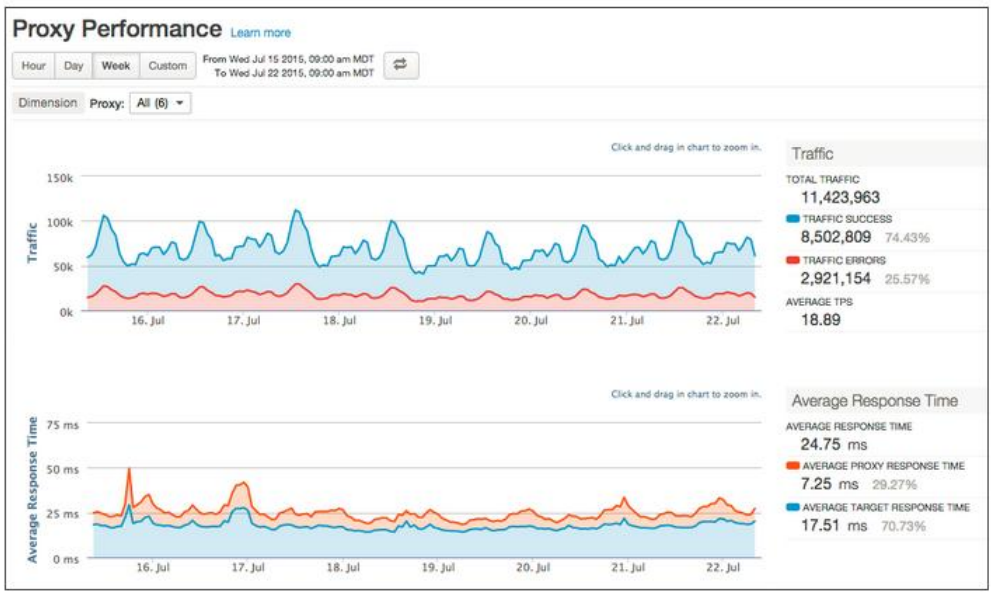
Metric	Description
	generating traffic (perhaps because their apps are still in development). If traffic starts later, in the most recent week, then calculated engagement will be higher for the most recent week than over the previous month, because most of the traffic was generated in the most recent week.
Developer Apps	Traffic for your top-performing developer apps for the selected time period is displayed in this chart.

Notice that the details chart also identifies and lists statistics for the APIs that have the most and least traffic and most and least errors over the selected period.

6.3.2.2 API proxy performance Dashboard

6.3.2.2.1 What does this dashboard tell me?

The API Proxy Performance dashboard tracks how much time API proxies spend processing requests and responses. This dashboard lets you visualize the network latency between SAP API Management and backend servers.



CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.3.2.2.2 What does this dashboard measure?

The API Proxy Performance dashboard tracks these metrics:

- Traffic
- Average Response Time
- Traffic by Proxy
- Average Response Time by Proxy

Traffic

Metric	Description
Total Traffic	The total number of API requests received by SAP API Management for an API environment in an organization.
Traffic Success	The total number of requests that resulted in a successful response. Error responses do not count.
Traffic Errors	The total number of all API requests that are unsuccessful, that is, the request does not deliver a response as desired by the end user.
Average TPS	The average nu

- Average Response Time

Metric	Description
Average Response Time	The average of the Total Response Time measured for all API calls made to an SAP API Management organization environment. The Total Response Time is the amount of time it takes for an API call to SAP API Management to return (in milliseconds). Or, put another way, total response time is the time measured from when an entire API call is received on SAP API Management to the time SAP API Management begins sending a response back to the client app. This chart measures the average for all proxies. For individual proxies, see the Average Response Time by Proxy chart below.
Average Proxy Response Time	This value is calculated as the average of the Total Response Time minus the Target Response Time for all API calls made to an SAP API Management organization environment. It's basically a measure of how much time the API calls spend flowing through SAP API Management itself (in milliseconds).
Average Target Response Time	The average number of milliseconds that it takes from the point the last byte of a request is sent from SAP API Management to a backend target to the time SAP API Management receives the last byte of the response. It's basically measuring how much time the API call spends on the target system.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Metric	Description
<Proxy name>	For the specified API proxy, the number of recorded API requests and responses.

- Traffic by Proxy

Metric	Description
<Proxy name>	For the specified API proxy, the number of recorded API requests and responses.

- Average Reponse Time by Proxy

Metric	Description
<Proxy name>	For the specified API proxy, the average of the Total Response Time measured for all API calls made to an SAP API Management environment. The Total Response Time is the amount of time it takes for an API call to SAP API Management to return (in milliseconds). Hover over the graph to see the total amount of time spent on the proxy side and the target side, as well as the average.

6.3.2.2.3 What else do I need to know about this dashboard?

You can view metrics for all proxies or drill into specific proxies using the Proxy dimension dropdown menu at the top of the dashboard.

Choose this icon to sort the selected metric:

- if you mouse over a light gray icon, it turns blue.
- A dark gray means it is currently sorted.

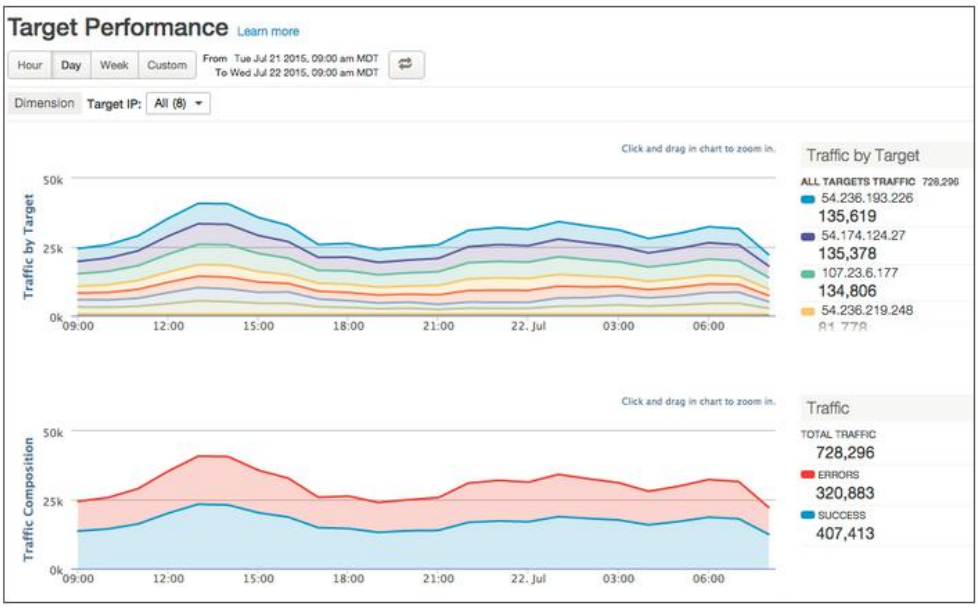
This dashboard uses standard controls, like the date and data aggregation selectors, hovering over graphs for more context, choose and drag to zoom, and so on.

This dashboard uses standard controls, like the date and data aggregation selectors, hovering over graphs for more context, and so on.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.3.2.3 Target Performance Dashboard

The Target Performance dashboard helps you visualize traffic patterns and performance metrics for API proxy backend targets.



6.3.2.3.1 What does this dashboard measure?

The API Proxy Performance dashboard tracks these metrics:

- Traffic by Target
- Traffic
- Response Time
- Target Errors
- Payload Size

- Traffic by Target

Metric	Description
All Targets Traffic	Measures the total amount of traffic that passes from SAP API Management to all backend targets.
<Target name>	Measures the total amount of traffic that passes from SAP API Management to the specified backend target.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- Traffic

Metric	Description
Total Traffic	Measures the total amount of traffic that passes from SAP API Management to all backend targets. Same as All Targets Traffic.
Errors	The total number of requests to backend targets that resulted in a successful response. Error responses do not count.
Success	The total number of all requests to backend targets that are unsuccessful (that do not return an error).

- Response Time

Metric	Description
Average Time	The average of the Total Response Time measured for all API calls made to an SAP API Management organization environment. The Total Response Time is the amount of time it takes for an API call to SAP API Management to return (in milliseconds). Or, put another way, total response time is the time measured from when an entire API call is received on SAP API Management to the time SAP API Management begins sending a response back to the client app.
Average Target Time	The average number of milliseconds that it takes from the point the last byte of a request is sent from SAP API Management to a backend target to the time SAP API Management receives the last byte of the response. It's basically measuring how much time the API call spends on the target system.
Average Proxy Time	This value is calculated as the Total Response Time minus the Target Response Time. It's basically a measure of how much time the API call spends flowing through SAP API Management itself (in milliseconds).

- Target Errors

Metric	Description
Total Errors	Measures the total number of errors sent from backend targets to SAP API Management.
3XX Errors	Measures the total number of HTTP 3XX sent from backend targets to SAP API Management.
4XX Errors	Measures the total number of HTTP 4XX errors sent from backend targets to SAP API Management.
5XX Errors	Measures the total number of HTTP 5XX sent from backend targets to SAP API Management.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- Payload Size

Metric	Description
Total Payload Size	The total payload size for all requests and responses between SAP API Management and backend targets.
Request Payload Size	The total payload size for all requests sent from SAP API Management to backend targets.
Response Payload Size	The total payload size for all responses sent from backend targets to SAP API Management.

6.3.2.3.2 What else do I need to know about this dashboard?

You can view metrics for all proxies or drill into specific proxies using the Target IP dimension dropdown menu at the top of the dashboard.

This dashboard uses standard controls, like the date and data aggregation selectors, hovering over graphs for more context, and so on.

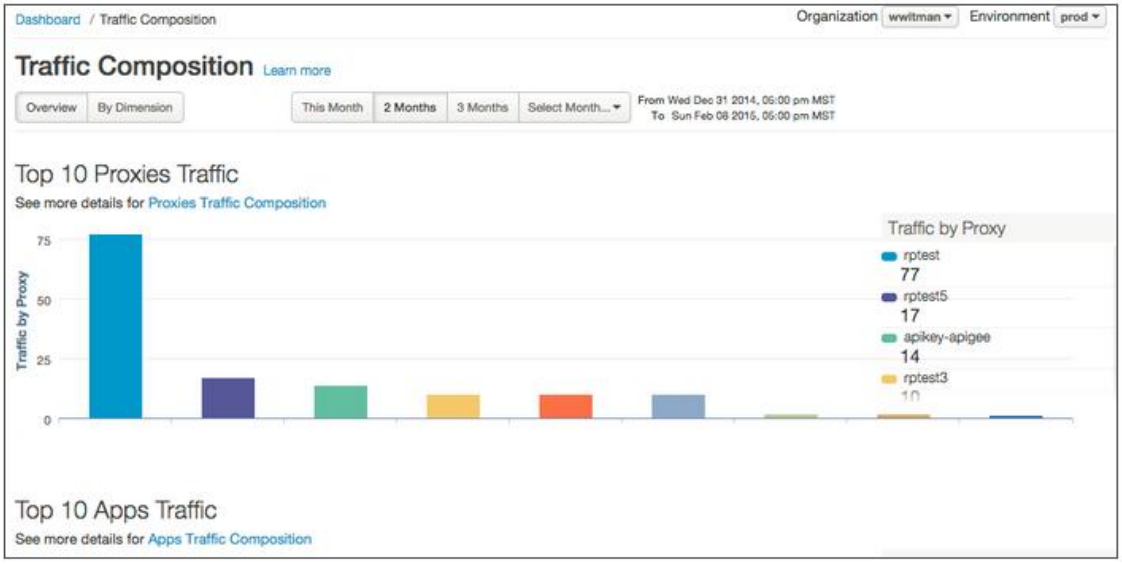
6.3.2.4 Traffic Composition Dashboard

6.3.2.4.1 What does this dashboard tell me?

The Traffic Composition dashboard measures the relative contribution of your top APIs, apps, developers, and products to your overall API program.

Use the report to detect business problems such as lower traffic trends or diminishing contribution from key apps and developers. Based on this data, you can decide what corrective action to take. You can also get early notification of new entities that contribute to API traffic, and take actions to respond — for example, you can determine which new developers are highly contributing and include them in nurturing programs.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.



6.3.2.4.2 What does this dashboard measure?

This dashboard has two measurement modes, which you select with the Overview and By Dimension buttons.

Metric	Description
Overview	Gives you a quick glance at traffic patterns for proxies, developer apps, products, and developers. For example, use this view to see which proxy, app, product, or developer is generating the most traffic.
By Dimension	Let's you drill in to see a more detailed view of proxy, developer app, product, and developer traffic.

6.3.2.4.3 What the Overview mode tells you

Choose Overview to display this mode. This mode presents a set of straight-forward graphs showing traffic patterns proxies, developer apps, developers, and products.

The following figure shows part of the traffic composition overview page:

Metric	Description
Top 10 Proxies Traffic	Shows the top 10 proxies measured by API traffic.
Top 10 Apps Traffic	Shows the top 10 developer apps measured by API traffic.
Top 10 Products Traffic	Shows the top 10 products measured by API traffic.
Top 10 Developers Traffic	Shows the top 10 developers measured by API traffic.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.3.2.4.4 What the By Dimension mode tells you

Choose By Dimension to display this mode.

Choose Overview to display this mode. This mode lets you drill in to see traffic pattern details for specific proxies, developer apps, developers, and products. For example, you can see at a glance how traffic for a specific proxy compares to all other proxies.

1. Select a dimension to examine. Dimensions include:

- Proxies
- Developer Apps
- Developers
- Products

2. Select the dimension instance to examine. For example, if the dimension is Developers, select a developer from the dropdown list, or select All to see metrics on all developers.

If you select All for a dimension, the dashboard offers charts that compare all entities together. For example, the Traffic by Dimension chart compares API proxy traffic for all API proxies in the organization.

Charts you will see in this mode include:

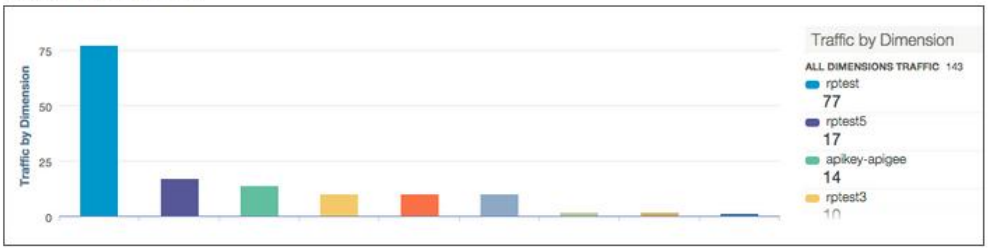
- Traffic composition
- Traffic by dimension
- Top 10 monthly composition

- Traffic composition

Shows the traffic for all entities of the selected dimension or all entities combined over the selected time interval.

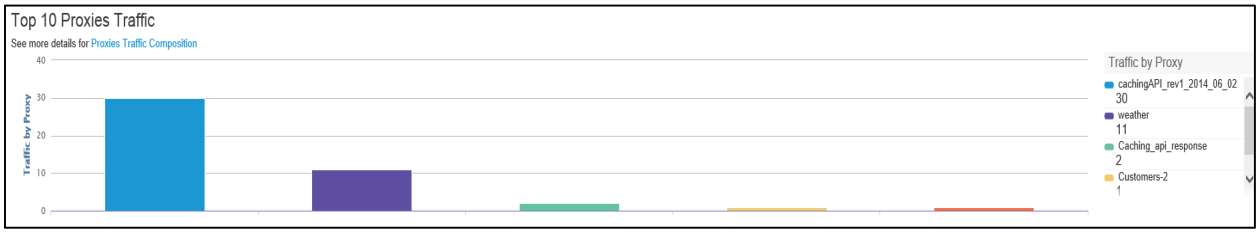
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Traffic by Dimension



Metric	Description
All Dimensions Traffic	Shows the traffic for all of the selected dimension in one chart. Dimensions can include API proxies, developer apps, developers, or products.
Proxy-specific traffic	Shows the traffic for the selected dimension.

Top 10 Proxies Traffic



6.3.2.4.5 What else do I need to know about this dashboard?

Use the dropdown menus to select (a) dimensions to examine (proxies, developer apps, developers, or products), and (b) specific entities to measure (either a specific entity like a proxy, or all proxies, for example).

This dashboard uses standard controls, like the date and data aggregation selectors, hovering over graphs for more context, and so on.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.3.2.5 Business Transaction Dashboard

6.3.2.5.1 What does this dashboard tell me?

The Business Transactions Dashboard lets you monitor and visually compare traffic for specific API patterns across multiple API proxies. This information helps you understand changes in API traffic that might be caused by specific business, marketing, or partner events.

For example, suppose that your marketing team hosts a program that drives customers to a specific site. The URI pattern for that site will experience an increase in traffic, giving the team a way to measure the impact of the program in real time.

6.3.2.5.2 What does this dashboard measure?

Metric	Description
Traffic	Also known as throughput. The number of API requests and resulting responses seen by the organization.
Average Response Time	The time an API takes to respond to an incoming request.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Metric	Description
Error Rate	The fraction of all API requests that are unsuccessful, that is, the request does not deliver a response as desired by the end user.
Average Data Exchange	The size of request and response. That is, the amount of data that is transferred in both directions as a request for an API and as a response that is generated and delivered to the calling entity.

6.3.2.5.3 How do I set up this dashboard?

The Business Transactions dashboard requires the following setup steps:

1. In the API Management menu, select API Proxies from the APIs menu.
2. In the API Proxies page, select the proxy you wish to measure.
3. In the Performance section of the proxy's Summary page,
4. In the Performance section of the page, you will see if there are any URI Patterns specified. To add a pattern, choose +URI Pattern. These are the patterns that the Business Transaction Dashboard will examine and compare with patterns from other APIs. For more details on configuring URI Patterns, see "Monitoring the performance of individual URIs in an API proxy".
5. From the Analytics menu, select Business Transaction Dashboard.
6. Choose the configuration (gear) icon next to the Business Transactions heading. The URI patterns for all APIs in your organization are listed there.
7. Select the URI patterns you wish to compare and specify a date range.
8. Choose Save.

To add more transactions, follow this same procedure to add URI patterns from as many APIs as you wish.

6.3.2.5.4 Monitoring the performance of individual URIs in an API proxy

Patterns begin with a forward slash and can include the asterisk (*) wildcard as part of the pattern.

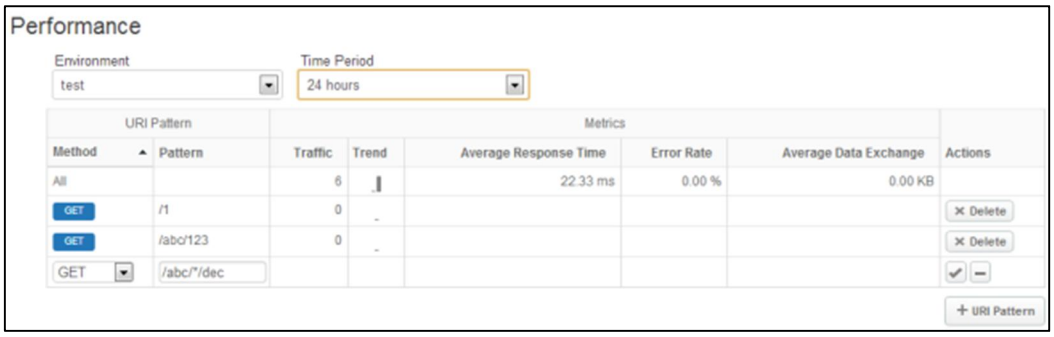
For example, given a proxy base path of /v1/inventory and resource paths of:

- /1
- /abc/123/dec
- /abc/456/dec

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

you can collect metrics on the following URIs:

- /1 - Collects metrics on the /1 URI
- /abc/123 - Collects metrics on the /abc/123 URI
- /abc/*/dec - Collects metrics on the /abc/123/dec and /abc/456/dec URIs



6.3.2.5.5 What else do I need to know about this dashboard?

This dashboard uses standard controls, like the date and data aggregation selectors, hovering over graphs for more context, and so on.

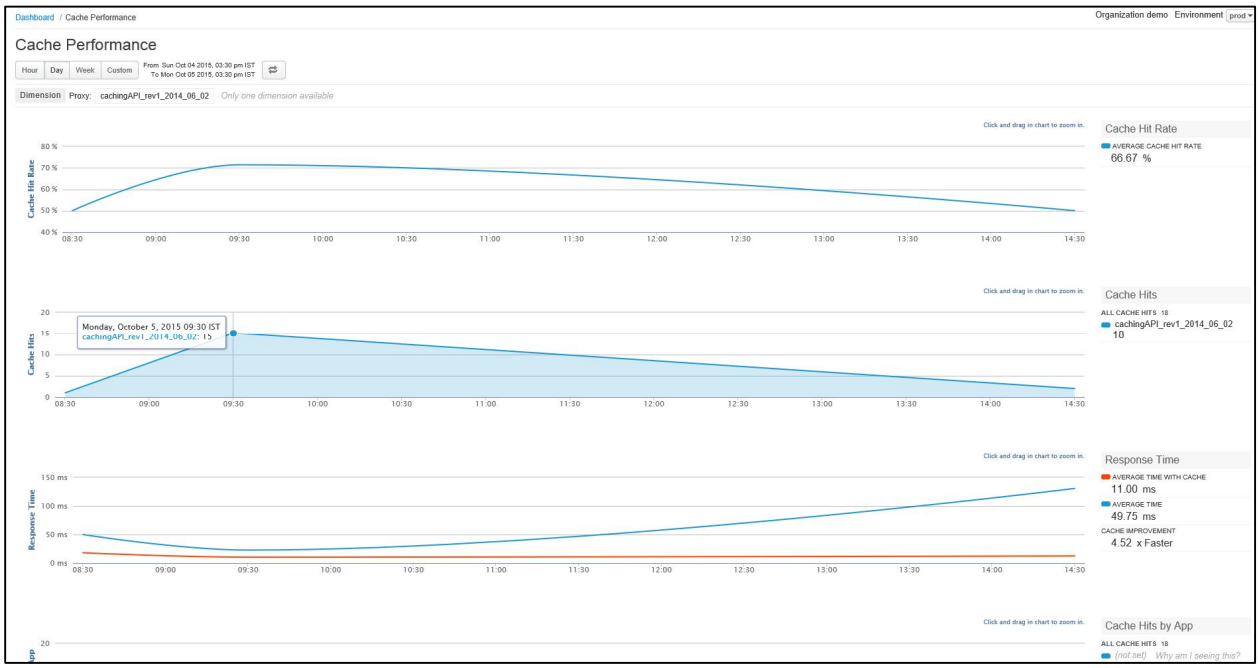
6.3.2.6 Cache performance dashboard

6.3.2.6.1 What does this dashboard tell me?

The Cache Performance dashboard lets you see at a glance the value of your SAP API Management cache. The dashboard helps you visualize the benefit of the cache in terms of lower latency and reduced load backend servers.

For information about cache management, see Manage caches for an environment.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.



6.3.2.6.2 What does this dashboard measure?

Metric	Description
Average cache hit rate	The rate of calls hitting the cache measured against total API traffic.
All cache hits	The sum of calls that hit the cache.
Average time with cache	The average amount of time for an API call when it hits the cache.
Average time without cache	The average amount of time for an API call when it does not hit the cache.
Cache improvement	Compares the average time with cache and average time without cache, giving you an idea of the overall effect of the cache on API performance.
Cache hits by app	Sum of calls hitting the cache broken down by developer app.

6.3.2.6.3 What else do I need to know about this dashboard?

You can measure cache performance for all API proxies in your organization, or you can select individual ones.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

This dashboard uses standard controls, like the date and data aggregation selectors, hovering over graphs for more context, and so on.

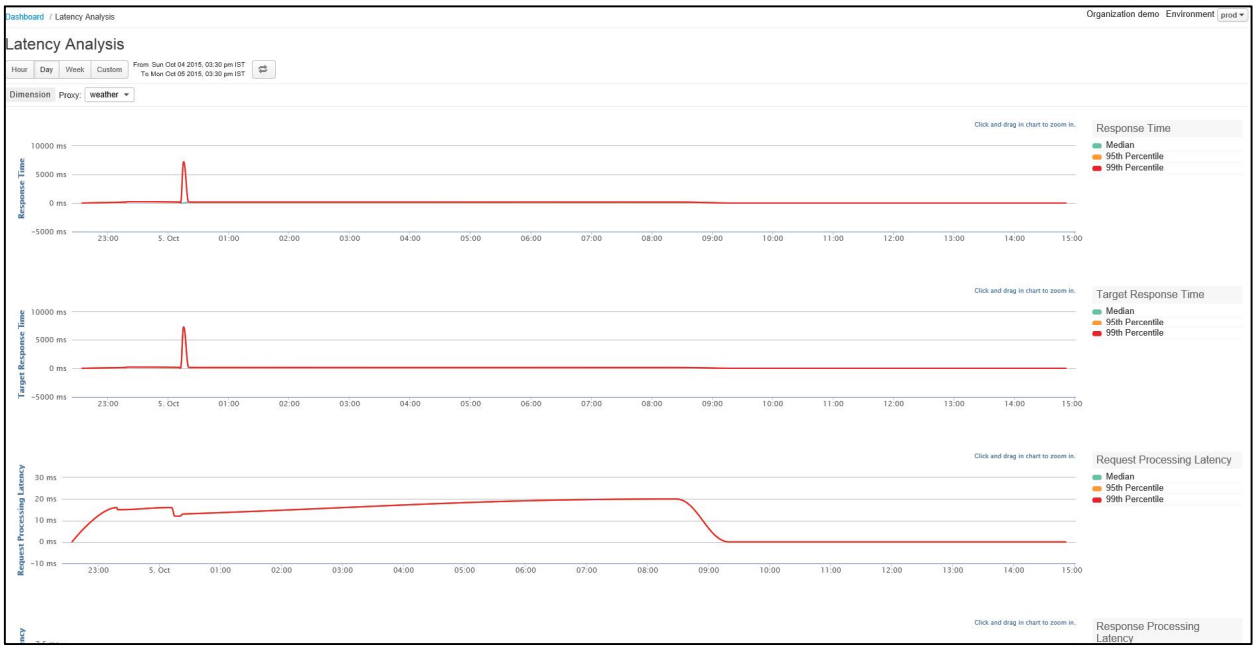
6.3.2.7 Latency Analysis Dashboard

6.3.2.7.1 What does this dashboard tell me?

The Latency Analysis dashboard can alert you to any latency issues your API proxies may be experiencing. It displays latency measurements down to the window of a minute, highlighting the median, 95th percentile, and 99th percentile values.

The median value tells you the point at which half of your traffic is experiencing latency that is less than this value and half of your traffic is experiencing latency that is greater than this value. For example, if the median response time latency for your selected API proxy is 62 ms, it means that half of the responses from this API proxy take less than 62 ms. It also means that half of the responses from this API proxy take more than 62 ms.

The 95th percentile and 99th percentile values tell you the point at which 95% and 99% of your traffic is experiencing latency that is less than these values. Or rather more importantly, it can point out outlying behavior, telling you that 5% and 1% of your traffic is experiencing latency values that are out-of-range.



© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.3.2.7.2 What does this dashboard measure?

Metric	Description
Response Time	Total number of milliseconds it took to respond to a call. This time includes the API proxy overhead and your target server time.
Target Response Time	Number of milliseconds it took your target server to respond to a call. This number tells you how your own servers are behaving.
Request Processing Latency	Number of milliseconds from the time when a call reaches the selected API proxy to the time when SAP API Management sends the call to your target server. Add the request and response latencies to calculate the final overhead the API proxy added to the call.
Response Processing Latency	Number of milliseconds from the time when the API proxy receives your target server's response to the time when SAP API Management sends the response to the original caller. Add the request and response latencies to calculate the final overhead the API proxy added to the call.

6.3.2.7.3 What else do I need to know about this dashboard?

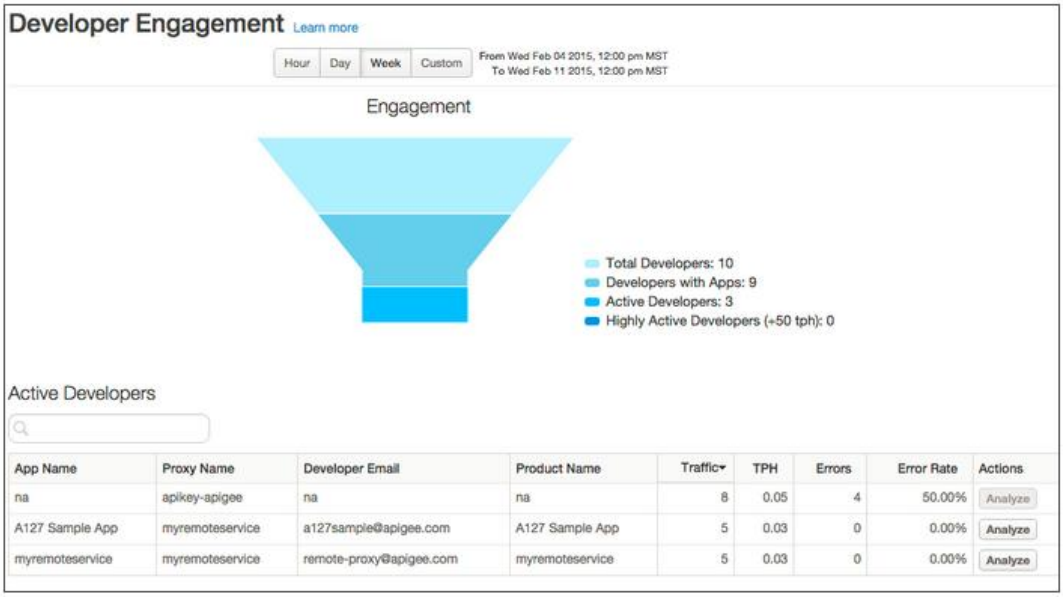
Select the API proxy for which you want to display latency values:

This dashboard uses standard controls, like the date aggregation selector, hovering over graphs for more context, and so on. To learn more, see Using the analytics dashboards section for more information.

6.3.2.8 Developer Engagement Dashboard

The Developer Engagement dashboard tells you which of your registered app developers are generating the most API traffic. For each of your developers, you can find out who is generating the most API traffic and the most errors. For example, if a particular developer's app is generating a lot of errors relative to other developers, you can pro-actively address the problem with that developer.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.



6.3.2.8.1 What does a custom report dashboard tell me?

The API Proxy Performance dashboard tracks these metrics:

- Engagement
- Active Developers
- Analyze

- Engagement

Metric	Description
Total Developers	The total number of developers associated with APIs deployed to an organization.
Developers with Apps	The total number of developers associated with apps in an organization.
Active Developers	The number of developers generating any amount of API traffic. Developers may exist in the organization and have apps, but if their apps are not making any API calls, they are not active.
Highly Active Developers	The number of developers generating API traffic exceeding 50 transactions per second.

- Active Developers

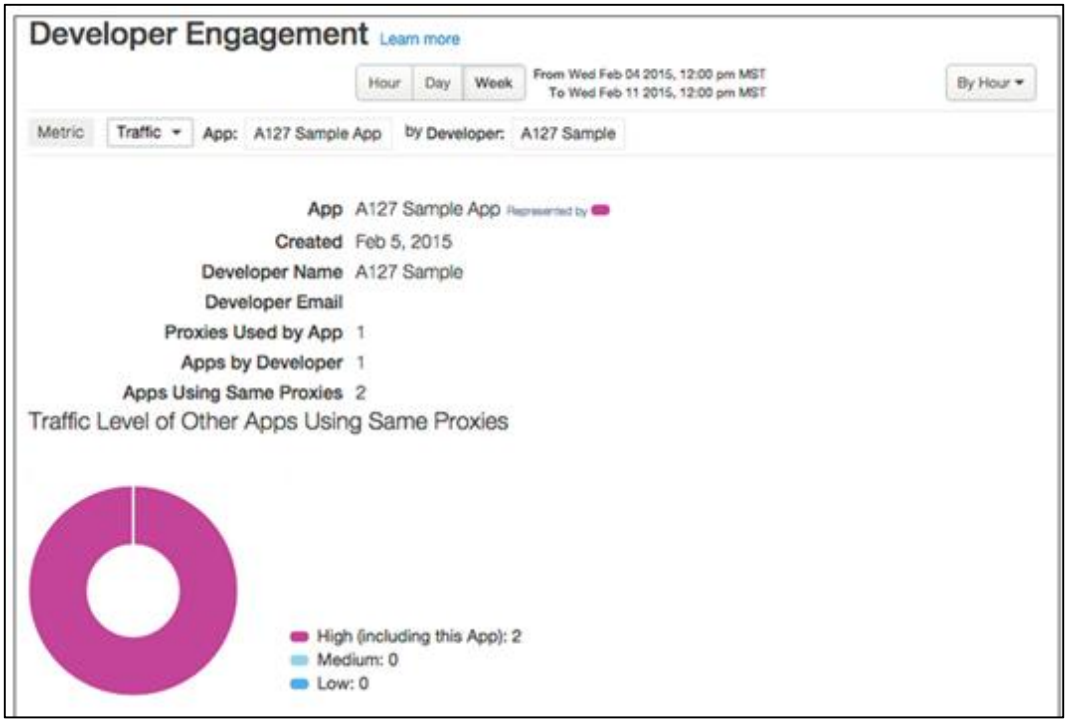
  "Active developers" means the number of developers generating any amount of API traffic. Developers may exist in the organization and have apps, but if their apps are not making any API calls, they are not active.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Metric	Description
App Name	The name of the app.
Proxy Name	The name of the API proxy associated with the app.
Developer Email	The email address of the developer who registered the app.
Product Name	The product name associated with the app.
Traffic	The amount of traffic generated by the app for the selected time period.
TPH	The transactions per hour generated by the app for the selected time period.
Errors	The total number of errors generated by the app for the selected time period.
Error Rate	The error percentage calculated by dividing total errors by total traffic for the selected time period.
Actions	Choose the Analyze button to see more details about an app.

Analyze developer engagement

In this view, select the metric you wish to view for the selected app.



CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Metric	Description
Proxies used by <App Name>	Measures all APIs that have traffic generated by the selected app.
Traffic level of other apps using the same proxies	Measures the API traffic flowing through all apps using the same set of APIs as the currently selected app.
Other Apps by Developer	Measures other apps generating traffic registered by the same developer as the developer of the selected app.
Other Apps using the same Proxies	Measures other apps, from the same developer or not, that use the same set of APIs used by the selected app.

6.3.2.8.2 What else do I need to know about this dashboard?

You can view metrics for all proxies or drill into specific proxies using the Metric dropdown menu at the top of the dashboard.

This dashboard uses standard controls, like the date and data aggregation selectors, hovering over graphs for more context, and so on

6.3.2.9 Custom Report

6.3.2.9.1 What does a custom report dashboard tell me?

A custom report dashboard is a dashboard that lets you visualize API analytics data from a previously created custom report. A custom report is a way to specify precisely what you want to measure across your API program. For instance, you can measure all API traffic generated from a specific client IP address.

Recommendation

You can arrange up to four custom report dashboards in a single custom reports dashboard.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

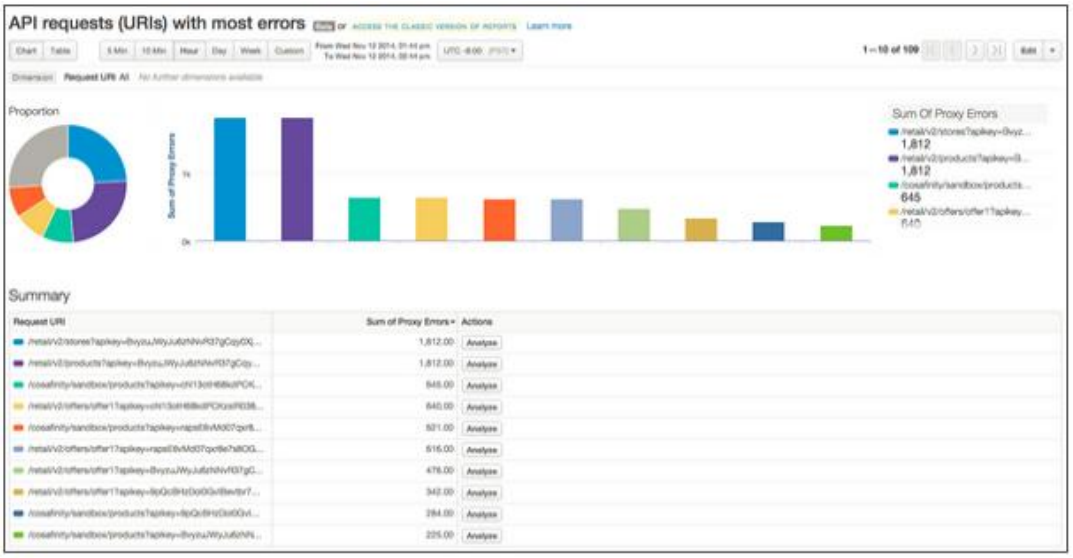
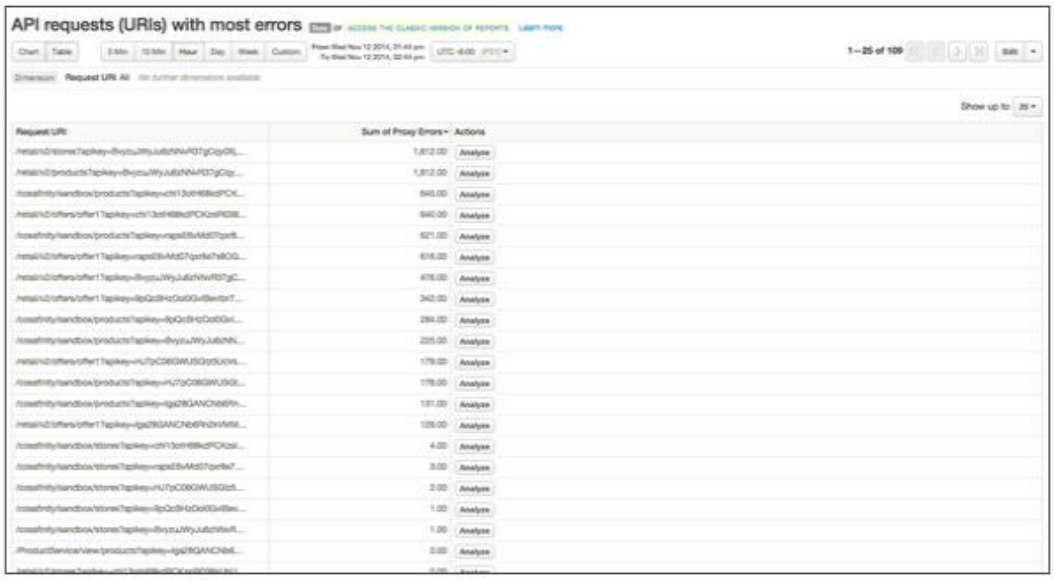


Table view



6.3.2.9.2 What does a custom report dashboard measure?

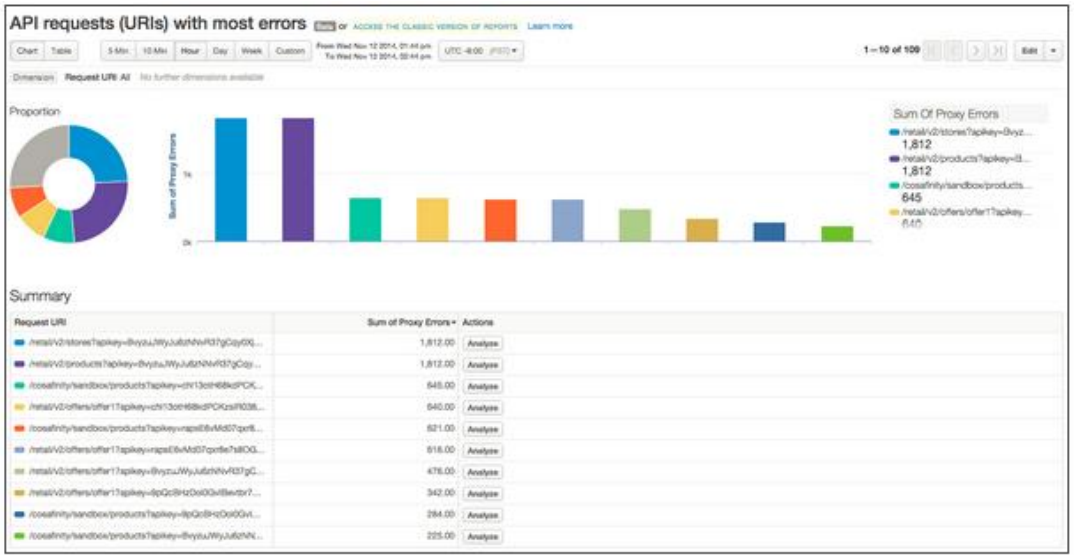
Custom report dashboards are similar in look and feel to other dashboards; however, the data that is displayed depends on which metrics and drilldown dimensions you selected when you created a corresponding custom report. See Custom reports dashboard for details.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.3.2.9.3 How do I view a custom report dashboard?

1. Choose Analytics in the main menu. The landing page lists all the available custom reports.

2. The Reports table lists the following elements:
   - Report name: Name of the report.
   - Environment: The environment in your org from which data is collected.
   - Metric: The primary measure specified for the report.
   - Report Description: The description of the report.
   - Last Modified: The date and time when the report was last modified.
3. Choose a report to view its custom dashboard.



6.3.2.9.4 What else do I need to know about custom reports dashboards?

- Dispersion charts

Shows the range of values over the selected time period for certain metrics.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- Analyze buttons

Choose Analyze in the Summary table to see more metrics, including Compare to Previous Period and Analyze Anomalies.

- Table view

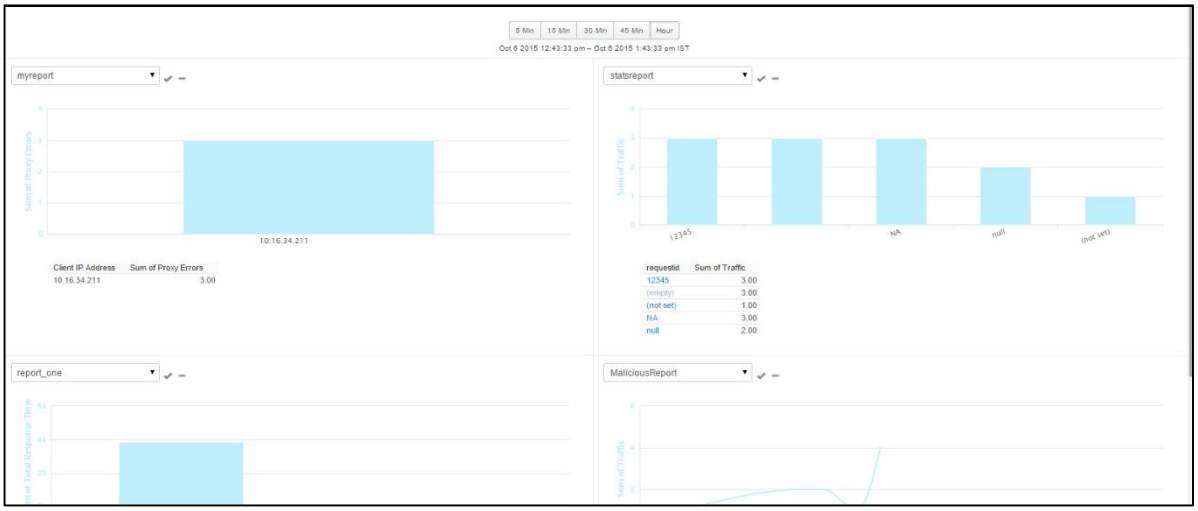
If your custom report includes many rows in the Summary table at the bottom of the Chart view, choose Table to access a user-friendly way to scroll through and display the Compare to Previous Period and Analyze Anomalies views of your custom report data.

Use the Show up to drop-down to control the number of rows you want to display in the table. You can display up to 200 rows at a time.

6.3.2.10 Custom Reports Dashboard

6.3.2.10.1 What does this dashboard tell me?

The Custom Reports dashboard lets you place up to four custom report dashboards on one page. This dashboard is intended to give you a "real time" view of your custom report metrics. The time-scale for this dashboard ranges from 5 minutes to one hour. The dashboard can include any custom reports that you previously created — including reports that use custom variables, drill-downs, metric correlations, and filters.



6.3.2.10.2 What does this dashboard measure?

The data displayed in this dashboard depends on the selected custom reports. You can pick from any custom reports that were previously created for your organization.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.3.2.10.3 What else do I need to know about this dashboard?

- Select Custom Reports Dashboard from the Analytics menu to view and configure the custom reports dashboard. Use the dropdown menus to select up to four custom reports to add to the dashboard. Choose the checkmark to add the selected report.

  > **Note**
  >
  > Custom dashboard configurations are persisted as cookies. This means that each user in your organization can have a different custom report dashboard instance. It also means that each user can maintain different instances from browser to browser.

- Choose a report name to display the full custom report dashboard. To change a report, select the Edit icon next to any report name:

6.3.3 Create Custom Reports

6.3.3.1 What you'll learn in this topic

After reading this topic, you will know how to:

- Use the Management UI to create custom reports using metrics and "drilldown" dimensions that you supply.
- Edit and save custom reports.
- Delete custom reports.
- Use the out-of-the-box custom reports.

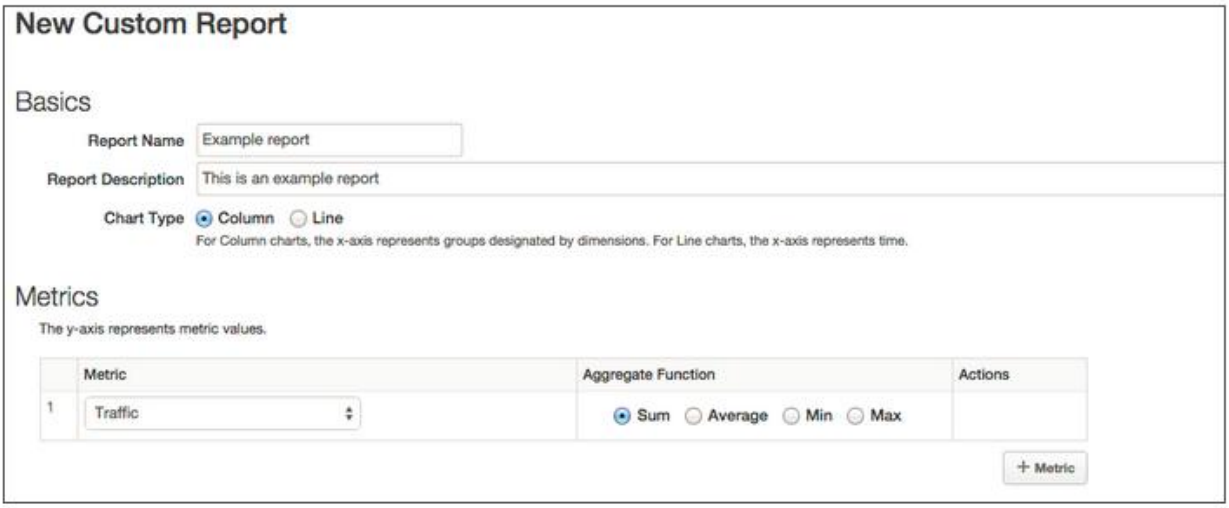
6.3.3.2 Adding new custom reports

By adding custom reports, you can create a set of charts that provide insight into every aspect of your API program.

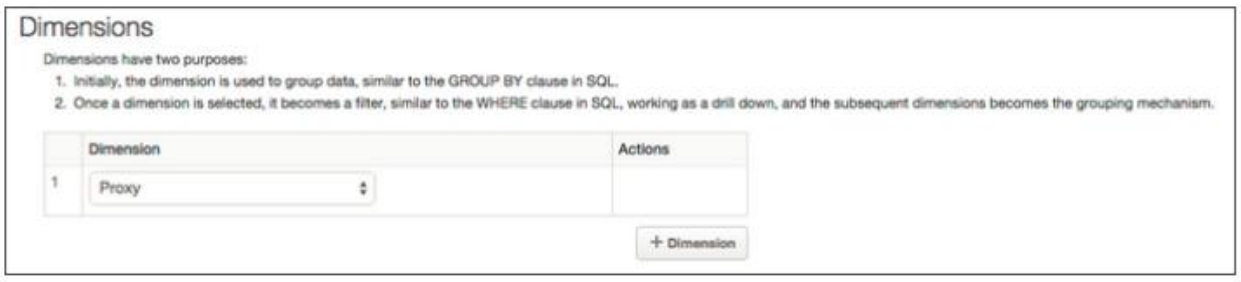
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

To add a custom report:

1. Select Custom Reports from the Analytics menu.
2. Choose + Custom Report.
3. Enter the name and description for your report.
4. Select a chart type, Column or Line. This is the style of chart that will be used to present your custom analytic data.
5. In the Measure sections, choose the metric that you wish to analyze. See "Custom metrics" for a description of each metric.
6. Select an aggregation function that you want applied to the data for the first metric. You can select an aggregation function to display the Sum, Average, Min, or Max values.
7. Choose + Measure to add additional measures.

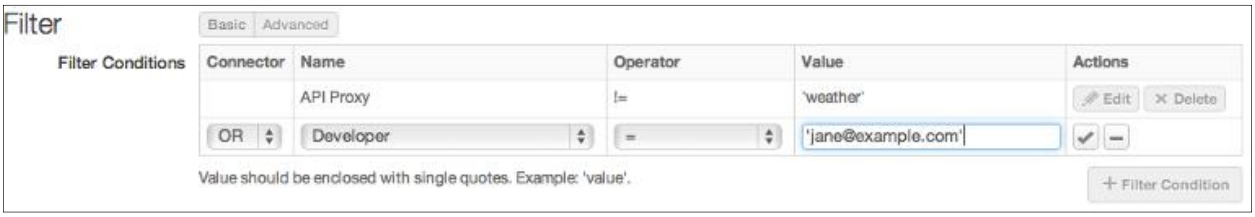


8. Choose your drilldown dimensions, such as an SAP API Platform API Proxy. Every dimension you add (by choosing + Drilldown) constrains the data set used to generate the reports. In effect, you are presenting more and more specific data with each drilldown.



9. You can further narrow the data displayed by adding filters to your report definition. In the Filter section of the page, choose + Filter Condition. Select the entity you want to filter on, and construct an expression with the Operator and Value to include or exclude data in the report. For example, you could add a filter that excludes data for the weather API proxy or developer jane@example.com.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.



Select an AND or OR connector for multiple filters, and choose the check mark icon to save each filter.

10. Choose Save.

You can view a list of all custom reports by selecting Analytics > Reports from the main menu.

With the report visible:

1. Select a data aggregation interval for the report. Note that the Hourly aggregation is the least costly choice in terms of system resources. Per Minute is the most costly.
2. Choose the Environment menu and choose the environment in your organization from which you want data to be collected.

6.3.3.3 Custom metrics

When building a custom report, you can choose from these metrics:

Metric	Description
Average transactions per second	The number of API requests and resulting responses per second.
Cache hit	The number of API requests that are serviced from the API platform cache. The request for the cached data is not forwarded to the backend target.
Policy errors	The request is marked as is_Error when it cannot be serviced successfully by a policy.
Proxy errors	The request is marked as is_Error when it cannot be serviced successfully by the proxy.
Request processing latency	The length of time it takes the proxy to complete an end-to-end transaction.
Request size	The size of the request in kilobytes.
Response processing latency	The time it takes for the proxy to process a request measured in milliseconds.
Response size	The size of the response in kilobytes.
Target errors	The request is marked as is_Error when it cannot be serviced successfully by the backend target.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Metric	Description
Target response time	The time it takes from when a request is sent from to the backend target and when it is received from the backend.
Total response time	The total time for an app to receive a response from an API request. This is the sum of the request processing latency of the proxy, request processing latency of the customer endpoint, response generation latency of the customer end point, and the response processing latency of the proxy. It also includes the latency introduced by the network layer as the request and response can go through multiple networks before it reaches its destination.
Traffic	Also known as throughput. The number of API requests and resulting responses seen by the organization.

6.3.3.4 Drilldown Dimensions

You can select from a large number of custom drilldown dimensions. Drilldown dimensions let you specify which dimensions to measure in your custom report. SAP API Management Analytics collects data on a wide range of dimensions. In addition, you can also select any custom dimensions that have been created in your organization. See Analytics reference.

6.3.3.5 Editing and Deleting your Reports

When you have completed building your custom report, the data is populated immediately. The graph is defaulted to plot the first metric you chose for the tab in your custom report. If you want to see different metrics, or compare them against each other, follow these steps:

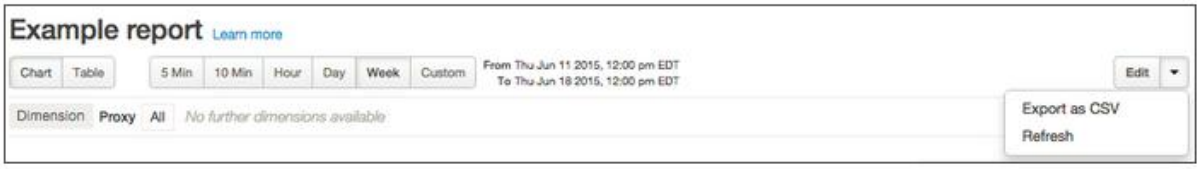
1. Choose the report name to view the graphs of the selected report.
2. Choose the Edit button to edit the display name, dimensions, and measures.
3. Choose Save.

You also have the option to delete a report by choosing the Delete button.

6.3.3.6 Exporting Saved Reports

You can export a saved custom report to CSV format. Each report includes an export menu where you can select a format for the export.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.



6.3.4 Use the analytics API to measure API program performance

This topic covers the SAP API Management Analytics Services API. SAP API Management records a wide variety of operational and business data that flows across APIs. The metrics derived from this data are useful for operational monitoring and business monitoring. Using Analytics Services, you can, for example, determine which APIs are performing well or poorly, which developers are delivering the highest value traffic, and which apps are causing the most issues for your backend services.

To help access this data easily, SAP API Management Analytics Services exposes a RESTful API. You can use this API when you need to automate certain Analytics functions, such as retrieving metrics periodically using an automation client or script. You can also use the API to build your own visualizations in the form of custom widgets that you can embed in portals or custom apps.

6.3.4.1 Retrieving statistics: dimensions, metrics, and functions

SAP API Management Analytics Services is based on the concept of 'dimensions'. A dimension is an entity that SAP API Management automatically monitors and measures. The measurements that Analytics collects are called 'statistics'. In the API, this is abbreviated as /stats.

For this reason, the base URL that you invoke to retrieve statistics for dimensions is the following:

```
<host>:<port>/v1/o/{org_name}/environments/{env_name}/stats
```

To create a working request, substitute your organization name for the variable {org_name}. Substitute the environment name ('test' or 'prod') for the variable {env_name}.

For example:

```
<host>:<port>/v1/o/apimakers/environments/prod/stats
```

This is the base URL that you invoke get statistics. To identify the dimension, you need to add the name of the dimension as a URI variable to this base URL.

The following dimensions are supported:

- /apis
- /apiproducts
- /apps
- /devs

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

For example, to get statistics for APIs:

```
<host>:<port>/v1/o/{org_name}/environments/{env_name}/stats/apis
```

To get statistics for developers:

```
<host>:<port>/v1/o/{org_name}/environments/{env_name}/stats/devs
```

To get statistics for developer apps:

```
<host>:<port>/v1/o/{org_name}/environments/{env_name}/stats/apps
```

To get statistics for APIs (or any other dimension) across all environments, a wildcard URI pattern is supported:

```
<host>:<port>/v1/o/{org_name}/environments/*/stats/apis
```

> **Recommendation**
>
> In this topic, you will get metrics for the set of built-in dimensions. The next topic, Analyze API message content using custom analytics, shows you how to define and work with custom dimensions.

You will find that if you submit one of these requests, you will get the following error:

```
{
  "code" : "Missing select parameter",
  "message" : "Missing select parameter",
  "contexts" : [ ]
}
```

Now you need to add a select statement to the base URL for the chosen dimension. You create the select statement using query parameters.

Query parameters define:

- Metrics: The attribute of the chosen dimension used to calculate statistics.
- Functions: The calculation run against the metric defined for the dimension.
- TimeRange: The time interval over which data should be analyzed.

> **Note**
>
> Data older than six months from the current date is not accessible by default. If you want to access data older than six months, contact SAP.

- Filters: The attribute used to drill-down into or sort across results.

> **Recommendation**
>
> For complete reference on these parameters, see Build custom reports.

Let us suppose that you want to get statistics across your API proxies that are deployed in the test environment. You want to find out how many request messages were received by all API proxies between 8/24/2013 and 9/25/2013. You want to find out a daily count for messages that were received. To do so, you create a select statement on the /stats/apis resource.

The example below demonstrates who you would construct a request for these statistics. The select statement defines the function sum for the metric message_count on the dimension apis deployed in the test environment.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The report provides a snapshot of the request message throughput to all APIs for traffic received between 8/24/2013 and 9/25/2013:

```
$ curl
<host>:<port>/v1/o/{org_name}/environments/test/stats/apis?"select=sum(message_coun
t)&timeRange=8/24/2013%2000:00~9/25/2013%2000:00&timeUnit=day" \
-u myname:mypass
```

Sample Response:

```
{
  "environments" : [ {
    "dimensions" : [ {
      "metrics" : [ {
        "name" : "sum(message_count)",
        "values" : [ {
          "timestamp" : 1379548800000,
          "value" : "1100.0"
        } ]
      } ],
      "name" : "target-reroute"
    } ],
    "name" : "test"
  } ],
  "metaData" : {
    "errors" : [ ],
    "failedEnvs" : "[]",
    "notices" : [ ],
    "samplingRate" : "100"
  }
```

This response indicates that 1100 message were received by one API proxy called 'target-reroute' running in the test environment between 8/24/2013 and 9/25/2013, on Thursday, 19 Sep 2013 00:00:00 GMT.

> **Note**
>
> The timestamp, 1379548800000, is in Unix timestamp format. There are many Unix timestamp conversion tools available on the Internet that you can use to decode these timestamps.

To get statistics for other dimensions, specify a different dimension as the URI parameter. For example, you can specify the apps dimensions to retrieve statistics for consumer apps. The following report shows the total throughput (messages received) from any apps for the specified time interval:

```
$ curl
<host>:<port>/v1/o/{org_name}/environments/test/stats/apps?"select=sum(message_coun
t)&timeRange=01/24/2013%2000:00~09/25/2013%2000:00&timeUnit=day" \
-u myname:mypass
--------------------------------------------------------------------------------
{
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
"environments" : [ {
  "dimensions" : [ {
    "metrics" : [ {
      "name" : "sum(message_count)",
      "values" : [ {
        "timestamp" : 1365379200000,
        "value" : "1.0"
      }, {
        "timestamp" : 1363046400000,
        "value" : "0"
      }, {
        "timestamp" : 1362441600000,
        "value" : "0"
      }, {
        "timestamp" : 1362355200000,
        "value" : "0"
      }, {
        "timestamp" : 1362096000000,
        "value" : "0"
      }, {
        "timestamp" : 1362009600000,
        "value" : "0"
      }, {
        "timestamp" : 1361404800000,
        "value" : "0"
      }, {
        "timestamp" : 1360022400000,
        "value" : "0"
      } ]
    } ],
    "name" : "weatherapp"
  }, {
    "metrics" : [ {
      "name" : "sum(message_count)",
      "values" : [ {
        "timestamp" : 1365379200000,
        "value" : "0"
      }, {
        "timestamp" : 1365033600000,
        "value" : "0"
      }, {
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
          "timestamp" : 1363305600000,
          "value" : "0"
        }, {
          "timestamp" : 1363046400000,
          "value" : "0"
        }, {
          "timestamp" : 1362441600000,
          "value" : "1.0"
        }, {
          "timestamp" : 1362355200000,
          "value" : "1.0"
        }, {
          "timestamp" : 1362096000000,
          "value" : "2.0"
        }, {
          "timestamp" : 1362009600000,
          "value" : "5.0"
        }, {
          "timestamp" : 1361404800000,
          "value" : "0"
        }, {
          "timestamp" : 1360022400000,
          "value" : "0"
        } ]
      } ],
      "name" : "thomas-app"
    } ],
    "name" : "test"
  } ],
  "metaData" : {
    "errors" : [ ],
    "failedEnvs" : "[]",
    "notices" : [ ],
    "samplingRate" : "100"
  }
}
```


Note

Statistics can be retrieved for apps only when apps can be individually identified by an API key (also called a 'consumer key') or by an OAuth access token. Therefore, only use the /apps dimension for APIs that enforce security policies (API Key or OAuth).

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.3.4.2 Using metrics

For each dimension, Analytics Services records pre-defined statistics for a set of relevant metrics. Metrics are attributes of dimensions that can be measured, such as total_response_time and request_size. In the examples above, you used the metric message_count, which os one of a set of metrics supported by the /apisdimension.

Metrics are specific to each dimension. For example, for the dimension /stats/devs, Analytics Services records the following metrics message_count, error_count, app_count, user_count. These provide statistics on the number of messages submitted by the developer's app(s), the number of errors generated by the developer's app(s), the number of apps the developer has registered, and the count of app end users associated with the developer's app(s).

Because these metrics are available out-of-the-box, they are referred to as "computed metrics" or "pre-aggregated metrics". Analytics Services provides you with a 'menu' of pre-computed metrics for each dimension. Since the metrics are already computed, you can retrieve large data sets very quickly.

Analytics Services also defines a set of dimensions that have no associated pre-computed metrics--for those dimensions, metrics are calculated on demand, when you make a request to Analytics Services.

Examples of dimensions whose statistics are computed on-demand are:

- `/stats/access_token`: Statistics based on OAuth access tokens presented by client apps.
- `/stats/client_id`: Statistics based on API keys presented by client apps.
- `/stats/client_ip`: Statistics based on the IP addresses of client apps.
- `/stats/target_url`: Statistics based on the URL of the target backend service.
- `/stats/request_verb`: Statistics based on the HTTP verb of the request message.

In this example, you use a dynamic dimension: target_url. Your goal is to determine how many requests to backend services result in errors. The report that you generate would be useful when communicating with the team responsible for backend services or when performing root-case analysis when an app developer reports issues when calling your APIs, and you suspect that the issue is related to failing backend services.

The `is_error` metric defines any message that results in an HTTP 4xx or 5xx HTTP status code. The sample request below defines the dimension target_url, the function count, and the metric is_error. The response will detail how many errors were returned by all target URLs called by API proxies deployed in the 'test' environment:

```
$ curl
<host>:<port>/v1/o/{org_name}/environments/test/stats/target_url?"select=count(is_e
rror)&timeRange=8/08/2013%2000:00~9/15/2013%2000:00&timeUnit=week \
-u myname:mypass
--------------------------------------------------------------------------------
{
  "environments" : [ {
    "dimensions" : [ {
      "metrics" : [ {
        "name" : "count(is_error)",
        "values" : [ {
          "timestamp" : 1344211200000,
          "value" : "44.0"
        }, {
          "timestamp" : 1344816000000,
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
          "value" : "1.0"
        } ]
      } ],
      "name" : "https://api.company.com"
    }, {
      "metrics" : [ {
        "name" : "count(is_error)",
        "values" : [ {
          "timestamp" : 1344211200000,
          "value" : "0"
        }, {
          "timestamp" : 1344816000000,
          "value" : "12.0"
        } ]
      } ],
      "name" : "http://weather.yahooapis.com"
    } ],
    "name" : "test"
  } ],
  "metaData" : {
    "samplingRate" : "100"
  }
```

Based on the statistics, you can see that the target URL `https://api.company.com` has a total of 45 errors during the period, while `http://weather.yahooapis.com` returned 12 errors during the period.

6.3.4.3 Sorting results by relative ranking

Many times when getting statistics, you only want to get results for a subset of the total set of metrics. Usually, you need to get the results for the "top 10", for example, the "top 10 slowest APIs", the "top 10 most active apps". You can do this using the topk query parameter as part of the request.

For example you may be interested to know who your top developers are, measured by throughput, or what your worst performers (i.e., 'top slowest') target APIs are by latency.

The topk (meaning 'top k' entities) enables reporting on the entities associated with the highest value for a given metric. This enables you to filter statistics for a list of entities that exemplify a particular condition. For example, to find which target URL was the most error prone over the last week, the topk parameter is appended to the request, with a value of 1.

In the example below

```
$ curl
<host>:<port>/v1/o/{org_name}/environments/test/stats/target_url?"select=count(is_error)&timeRange=8/08/2013%2000:00~8/15/2013%2000:00&timeUnit=week&sortby=count(is_error)&topk=1" -u myname:mypass
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
{
  "environments" : [ {
    "dimensions" : [ {
      "metrics" : [ {
        "name" : "count(is_error)",
        "values" : [ {
          "timestamp" : 1344816000000,
          "value" : "1.0"
        }, {
          "timestamp" : 1344211200000,
          "value" : "44.0"
        } ]
      } ],
      "name" : "https://api.company.com"
    } ],
    "name" : "test"
  } ],
  "metaData" : {
    "samplingRate" : "100"
  }
}
```

The result of this request is a set of statistics that show that the buggiest target URL is `http://api.company.com`.

You can also use the `topk` parameter to sort for the APIs experiencing the highest throughput. The following example retrieves statistics on the top ranked API, defined by highest throughput in the last week:

```
$ curl
<host>:<port>/v1/o/{org_name}/environments/test/stats/apis?"select=message_count&timeRange=7/24/2012%2000:00~8/25/2012%2000:00&timeUnit=week&sortby=message_count&sort=DESC&topk=1" \
-u myname:mypass
```

Sample Response

```
{
  "environments" : [ {
    "dimensions" : [ {
      "metrics" : [ {
        "name" : "message_count",
        "values" : [ {
          "timestamp" : 1345420800000,
          "value" : "14.0"
        }, {
          "timestamp" : 1344816000000,
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
          "value" : "8.0"
        }, {
          "timestamp" : 1344211200000,
          "value" : "111.0"
        }, {
          "timestamp" : 1343606400000,
          "value" : "405.0"
        } ]
      } ],
      "name" : "weatherAPI"
    } ],
    "name" : "test"
  } ],
  "metaData" : {
  }
}
```

6.3.4.4 Filtering results

For greater granularity, you can filter results to enable 'drill-down' on a dimension of interest. When using filters, you must use dynamic dimensions as filter properties. Computer metrics (/apis, /devs, /apps, etc.) cannot be used as filters.

For example, let's suppose that you need to retrieve a count of errors from backend services filtered by the HTTP verb of the request. Your goal is find out how many POST and PUT requests are generating errors per backend service. To do so, you use the dimension target_url along with the filter request_verb.

```
$ curl
<host>:<port>/v1/o/{org_name}/environments/test/stats/target_url?"select=count(is_error)&timeRange=8/08/2013%2000:00~8/15/2013%2000:00&timeUnit=week&filter=(request_verb&20in&20'POST','PUT')" \
-u myname:mypass
```

6.3.4.5 Paginating results

In production environments, some request to the Analytics Services API return very large data sets. To make it easy to display large data sets in the context of a UI-based application, the API natively supports pagination.

To paginate results, use the `offset` and `limit` query parameters.

For example, the following request would be likely to return a large data set, since it retrieves metrics for all errors on all APIs in the product environment for the last week.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
$ curl
<host>:<port>/v1/o/{org_name}/environments/prod/stats/apis?"select=count(is_error)&
timeRange=8/08/2013%2000:00~8/15/2013%2000:00&timeUnit=week" \
-u myname:mypass
```

If your UI-based application can reasonably display 50 results per page, you can set the limit to 50, as follows:

```
$ curl
<host>:<port>/v1/o/{org_name}/environments/prod/stats/apis?"select=count(is_error)&
timeRange=8/08/2013%2000:00~8/15/2013%2000:00&timeUnit=week&limit=50&offset=0" \
-u myname:mypass
```

For the second 'page' of results, use the offset query parameter, as follows:

```
$ curl
<host>:<port>/v1/o/{org_name}/environments/prod/stats/apis?"select=count(is_error)&
timeRange=8/08/2013%2000:00~8/15/2013%2000:00&timeUnit=week&limit=50&offset=1" \
-u myname:mypass
```

6.3.4.6 Reference: Analytics calls made by the API dashboard

The following API calls are made by the dashboards in the Management UI. They are provided here for your reference.

6.3.4.7 All API Traffic

```
$ curl
<host>:<port>/v1/o/{org_name}/environments/test/stats/?'select=message_count&timeRa
nge=7/14/2012%2013:00~8/13/2012%2013:00&sort=ASC&sortby=message_count&timeUnit=hour
' -u myname:mypass
```

6.3.4.8 Top API Movers

```
$ curl
'<host>:<port>/v1/o/{org_name}/environments/test/stats/apis?select=message_count,er
ror_count,&timeRange=7/14/2012%2013:00~8/13/2012%2013:00&sort=ASC&topk=5&sortby=mes
sage_count&timeUnit=hour' -u myname:mypass
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.3.4.9 Top API Products

```
$ curl
'<host>:<port>/v1/o/{org_name}/environments/test/stats/apiproducts?select=message_c
ount,app_count,developer_count&timeRange=7/14/2012%2013:00~8/13/2012%2013:00&sort=A
SC&topk=5&sortby=app_count&timeUnit=hour' -u myname:mypass
```

6.3.4.10 Top Ranked Apps

```
$ curl
'<host>:<port>/v1/o/{org_name}/environments/test/stats/apps?select=message_count,er
ror_count,user_count&timeRange=7/14/2012%2013:00~8/13/2012%2013:00&sort=ASC&topk=5&
sortby=user_count&timeUnit=hour' -u myname:mypass
```

Top Ranked Developers

```
$ curl
'<host>:<port>/v1/o/{org_name}/environments/test/stats/devs?select=message_count,er
ror_count,user_count&timeRange=7/14/2012%2013:00~8/13/2012%2013:00&sort=ASC&topk=5&
sortby=user_count&timeUnit=hour' -u myname:mypass
```

6.3.4.11 Opt in or out of daily analytics reports

In addition to the analytics available online through SAP API Management, all organization administrators are automatically subscribed to receive daily analytics summary reports through email. You can unsubscribe (opt out) of receiving daily analytics reports through a link in the actual report that is emailed to you. You can also opt in or opt out of daily summary reports using these API calls:

Opt out of receiving a daily summary report:

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/stats/preferences/reports/dailysummaryreport?optin=false
```

Opt in to receiving a daily summary report:

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/stats/preferences/reports/dailysummaryreport?optin=true
```

Retrieve list of users who have opted in or opted out:

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}stats/preferences/reports/dailysummaryreport/users
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.3.5 Analyze API message content using custom analytics

6.3.5.1 Introduction

This topic presents a tour demonstrating how to use policies to extract custom data from a request and feed that data to the SAP API Platform Analytics system. Then, we show how to create a custom analytics report based on that data, which appears as custom dimensions. In addition, this topic explains how to create custom analytics variables (dimensions) using a UI tool called Solution Builder.

6.3.5.2 Extracting custom analytics data using policies

In Use the analytics API to measure API program performance, you learned how to use the RESTful API exposed by Analytics Services to get statistics on a variety of entities monitored by SAP API Management.

In this topic, you will learn how to use the Analytics Services API combined with Policies to analyze data that is unique to your app and API traffic. Most of the data that is key to your business is found in the payload content moving back and forth from apps to your backend services. Using Analytics Services, you can define custom dimensions that SAP API Management uses to collect, analyze, and provide reports on that data.

This topic demonstrates the usage of custom analytics against the Yahoo Weather API. The goal of the exercise is to create a custom dimension called location that enables you to collect statistics on the number of requests received for weather reports for different locations. Once you have defined the custom dimension called location, you can use the capabilities of the Analytics Services API to retrieve and filter statistics that the SAP API Management collects. The custom dimension location will also be made available in the Management UI, enabling you to create reports for this dimension using the API SAP API Platform's UI-based tools.

6.3.5.2.1 Parsing payloads using Policies

The Yahoo Weather API returns XML-formatted responses. You request a weather report for particular location by providing a WOEID, which stand for "where on Earth ID". The WEOID for Palo Alto, CA is 12797282. To get a weather forecast for Palo Alto, you submit the following request to the Yahoo Weather API:

```
$ curl http://weather.yahooapis.com/forecastrss?w=12797282
```

To collect custom analytics, you need to call the API using an API proxy. The API proxy will inspect the request and response messages from your app to the Yahoo API. You are provided with a pre-configured API proxy in the test environment of your organization. The API proxy is called weatherapi. You can invoke that API proxy to obtain a proxied response from the the Yahoo Weather API.

You can invoke the proxied version of the Yahoo Weather API using the following command. As usual, substitute your organization name on SAP API Management for the variable {org_name}.

```
$ curl http://{org-name}-{env}.<host:port>/weather/forecastrss?w=12797282
```

The interesting part of the response message, the weather report and forecast, is shown below. (Note that the response, except for specifics such as timestamps, is exactly the same between the direct API call to the weather backend and the proxied API call.)

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<rss version="2.0" xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">
 <channel>
  <item>
   <!-- Some XML excluded here. . . for brevity -->
   <yweather:forecast day="Wed" date="1 Oct 2014" low="49" high="72" text="Sunny"
code="30" />
   <yweather:forecast day="Thu" date="2 Oct 2014" low="48" high="73" text="Sunny"
code="30" />
   <yweather:forecast day="Fri" date="3 Oct 2014" low="47" high="72" text="Sunny"
code="32" />
   <yweather:forecast day="Sat" date="4 Oct 2014" low="48" high="75" text="Sunny"
code="32" />
   <yweather:forecast day="Sun" date="5 Oct 2014" low="49" high="77" text="Sunny"
code="32" />
   <guid isPermaLink="false">USCA1093_2014_1_13_7_00_PDT</guid>
   </item>
 </channel>
</rss>
```

To see the same data for a different location, submit the same request with a different WOEID, 2520841, for Williamsburg.

```
$ curl http://{org-name}-{env}.<host:port>/weather/forecastrss?w=2520841
```

ExtractVariables policy

The weather response contains potentially valuable information. However, SAP API Management doesn't yet 'know' how to feed this message content into Analytics Services for processing. To enable this, SAP API Platform provides the ExtractVariables policy, which parses message payloads with JSONPath or XPath expressions and feeds the content into Analytics Services.

There are many tools available online that you can use to construct XPath expressions for your XML documents. There also many tools available for JSONPath.

See Extract message content using ExtractVariables.

To extract the information of interest from the weather report in this case, you use an XPath expression. For example, to extract the value of the city, the XPath expression is:

```
/rss/channel/yweather:location/@city
```

Note how this XPath expression reflects the structure of the XML nodes. also, note the prefixyweather is defined by a namespace:

```
xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0
```

To enable the XML message to be parsed properly, you use both the XPath and the namespace definition in the policy.

You will include this XPath expression in the ExtractVariables Policy. Once the XPath has been evaluated, the Policy needs a place to store the value that results from the evaluation. For this, the Policy uses variables. You can

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

create custom variables whenever you need them by defining a variable prefix and variable name in the ExtractVariables Policy. The Policy can then store the value that it finds by parsing the message payload in the variable, and Policies and code can retrieve that value to do some additional processing on it.

In this example, you define four custom variables:

- `weather.location`
- `weather.condition`
- `weather.forecast_today`
- `weather.forecast_today`

In these variables that you define here, weather is a prefix, and location, condition, forecast_today, and forecast_today are each variable names, and each will have an associated XPath expression.

Naming restrictions

See the naming restrictions for custom analytics variables in the Creating custom analytics variables with the Solution Builder section.

- The following naming restrictions apply to custom analytics variables:
- Names cannot be multiple-word (no spaces).
- No quotation marks, underscores, hyphens, or periods.
- No special characters.

Example

The following ExtractVariables Policy configuration demonstrates how you can define these custom variables. Note that the VariablePrefix is weather, and that each <Variable> tag has a name attribute that maps to the list of custom variables that you need to have populated. Each variable in turn has an associated XPath expression that will populate the variable with the appropriate value for each response message.

To create the Policy, under `/apiproxy/policies`, create a file called `ParseWeatherReport.xml` with the following content:

```
<ExtractVariables name="ParseWeatherReport">
 <!-- Parse the XML weather report using XPath. -->
 <VariablePrefix>weather</VariablePrefix>
  <XMLPayload>
   <Namespaces>
    <Namespace
prefix="yweather">http://xml.weather.yahoo.com/ns/rss/1.0</Namespace>
   </Namespaces>
   <Variable name="location" type="string">
    <XPath>/rss/channel/yweather:location/@city</XPath>
   </Variable>
   <Variable name="condition" type="string">
    <XPath>/rss/channel/item/yweather:condition/@text</XPath>
   </Variable>
   <Variable name="forecast_today" type="string">
    <XPath>/rss/channel/item/yweather:forecast[1]/@text</XPath>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    </Variable>
    <Variable name="forecast_tomorrow" type="string">
     <XPath>/rss/channel/item/yweather:forecast[2]/@text</XPath>
    </Variable>
  </XMLPayload>
</ExtractVariables>
```

StatisticsCollector policy

The next step is to create another Policy that reads these values and sends them to Analytics Services for processing. The StatisticsCollector Policy type is used to do this. In the StatisticsCollector Policy, you define a Statistic by providing a pointer to each of the custom variables defined in the ExtractVariables Policy. You can also provide a default value for custom variable, which will be forwarded to Analytics Services if the variables cannot be resolved on a response. (In the example below, the default values are Earth, Sunny, Rainy, and Balmy.)

Under `/apiproxy/policies`, create a file called `AnalyzeWeatherReport.xml` with the following content:

```
<StatisticsCollector name="AnalyzeWeatherReport">
 <Statistics>
  <Statistic name="location" ref="weather.location" type="string">Earth</Statistic>
  <Statistic name="condition" ref="weather.condition"
type="string">Sunny</Statistic>
  <Statistic name="forecast_today" ref="weather.forecast_today"
type="string">Rainy</Statistic>
  <Statistic name="forecast_tomorrow" ref="weather.forecast_tomorrow"
type="string">Balmy</Statistic>
 </Statistics>
</StatisticsCollector>
```

6.3.5.2.2 Attaching policies to the ProxyEndpoint response Flow

To make things work properly, Policies must be attached to the API proxy Flow in the appropriate location. In this use case, the Policies must execute after the response has been received from the Yahoo Weather API and before the response is sent to the request client. To accomplish this, the Policies must be attached to the ProxyEndpoint response Flow, so that they will be enforced on outbound response messages, before the response is returned to the calling client app.

The example ProxyEndpoint configuration below first executes the Policy called 'ParseWeatherReport' to parse the response message. The ParseWeatherReport evaluates the XPath expressions and populates appropriate variables. The policy called 'AnalyzeWeatherReport' then forwards those values to Analytics Services.

```
<ProxyEndpoint name="default">
 <Flows>
  <Flow name="default">
   <Response>
    <Step><Name>ParseWeatherReport</Name></Step>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <Step><Name>AnalyzeWeatherReport</Name></Step>
   </Response>
  </Flow>
 </Flows>
 <HTTPProxyConnection>
  <!-- Base path used to route inbound requests to this API proxy -->
  <BasePath>/weather</BasePath>
  <!-- The named virtual host that defines the base URL for requests to this proxy
-->
  <VirtualHost>default</VirtualHost>
 </HTTPProxyConnection>
 <RouteRule name="default">
 <!-- Connects the proxy to the target defined under /targets -->
  <TargetEndpoint>default</TargetEndpoint>
 </RouteRule>
</ProxyEndpoint>
```

6.3.5.2.3 Importing and deploying the API proxy

After you have made these changes, you need to import and deploy the API proxy that you have configured.

6.3.5.2.4 Populating Analytics data for custom variables

After you deploy your changes, you need to populate some data in Analytics Services. You can do this by running the following commands, each of which uses a WOEID for a different geographic location.

Palo Alto:

```
$ curl http://{org-name}-{env}.<host:port>/weather/forecastrss?w=12797282
```

Shanghai:

```
$ curl http://{org-name}-{env}.<host:port>/weather/forecastrss?w=2151849
```

London:

```
$ curl http://{org-name}-{env}.<host:port>/weather/forecastrss?w=44418
```

Wiliamsburg:

```
$ curl http://{org-name}-{env}.<host:port>/weather/forecastrss?w=2520841
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.3.5.2.5 Generating a custom report

Now that you have populated some data, you can use the RESTful API exposed by Analytics Services to get statistics on your new custom dimensions, in the same way that you use the API to get statistics on the out-of-the-box dimensions.

You can also generate custom reports using the Management UI at. You will find that a new dimension called location is available in the UI for you to use in generating reports.

> **Note**
>
> `TimeRange` parameter must be modified to include the time interval when data was collected.

In the example request below, the custom dimension is called location. This request builds a custom report for locations based on the sum of message counts submitted for each location.

As usual, substitute your organization name for the variable {org_name}, and substitute the username and password for your account on SAP API Management for myname:mypass:

```
$ curl
<host>:<port>/v1/o/{org_name}/environments/test/stats/location?"select=sum(message_
count)&timeRange=11/19/2012%2000:00~11/21/2012%2000:00&timeUnit=day"
-u myname:mypass
--------------------------------------------------------------------------------
{
  "environments" : [ {
    "dimensions" : [ {
      "metrics" : [ {
        "name" : "sum(message_count)",
        "values" : [ {
          "timestamp" : 1353369600000,
          "value" : "4.0"
        } ]
      } ],
      "name" : "London"
    }, {
      "metrics" : [ {
        "name" : "sum(message_count)",
        "values" : [ {
          "timestamp" : 1353369600000,
          "value" : "19.0"
        } ]
      } ],
      "name" : "Palo Alto"
    }, {
      "metrics" : [ {
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        "name" : "sum(message_count)",
        "values" : [ {
          "timestamp" : 1353369600000,
          "value" : "2.0"
        } ]
      } ],
      "name" : "Shanghai"
    }, {
      "metrics" : [ {
        "name" : "sum(message_count)",
        "values" : [ {
          "timestamp" : 1353369600000,
          "value" : "14.0"
        } ]
      } ],
      "name" : "Williamsburg"
    } ],
    "name" : "test"
  } ],
  "metaData" : {
    "samplingRate" : "100"
  }
}
```

In some cases, there may be a large number of results. It may be useful to filter the list to report the top 2 locations by message volume. This is done by adding the topk query parameter and providing an integer value for the number to filter:

```
$ curl
<host>:<port>/v1/o/{org_name}/environments/test/stats/location?'select=sum(message_
count)&timeRange=11/19/2012%2000:00~11/21/2012%2000:00&timeUnit=day&sortby=sum(mess
age_count)&topk=2" \
-u myname:mypass
```

Results can also be filtered by specifying the values of the dimensions of interest. In the example below, the report is filtered by results for London and Shanghai :

```
$ curl
<host>:<port>/v1/o/{org_name}/environments/test/stats/location?"select=sum(message_
count)&timeRange=11/19/2012%2000:00~11/21/2012%2000:00&timeUnit=day&filter=(locatio
n%20in%20'London','Shanghai')" \
-u myname:mypass
--------------------------------------------------------------------------------
{
  "environments" : [ {
    "dimensions" : [ {
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
      "metrics" : [ {
        "name" : "sum(message_count)",
        "values" : [ {
          "timestamp" : 1353369600000,
          "value" : "4.0"
        } ]
      } ],
      "name" : "London"
    }, {
      "metrics" : [ {
        "name" : "sum(message_count)",
        "values" : [ {
          "timestamp" : 1353369600000,
          "value" : "2.0"
        } ]
      } ],
      "name" : "Shanghai"
    } ],
    "name" : "test"
  } ],
  "metaData" : {
    "samplingRate" : "100"
  }
}
```

6.3.5.3 Creating custom analytics variables with the Solution Builder

Solution builder lets you create custom analytics variables through an easy-to-use management UI dialog.

> **Note**
>
> You may wish to read the previous section "Parsing payloads using Policies", which explains how the ExtractVariables and StatisticsCollector policies work hand-in-hand to feed custom variables to the Analytics system. As you'll see, the UI follows this same pattern, but provides a convenient way for you to configure things entirely through the management UI. If you wish, try the Yahoo Weather API tutorial presented previously in this topic using the UI instead of editing and attaching policies manually.

> **Caution**
>
> Naming restrictions for custom analytics variables
>
> - The following naming restrictions apply to custom analytics variables:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- Names cannot be multiple-word (no spaces).
- No quotation marks, underscores, hyphens, or periods.
- No special characters.
- In addition to the above, follow the column naming restrictions here: http://www.postgresql.org/docs/8.0/static/sql-syntax.html#SQL-SYNTAX-IDENTIFIERS

Using the Solution Builder dialog

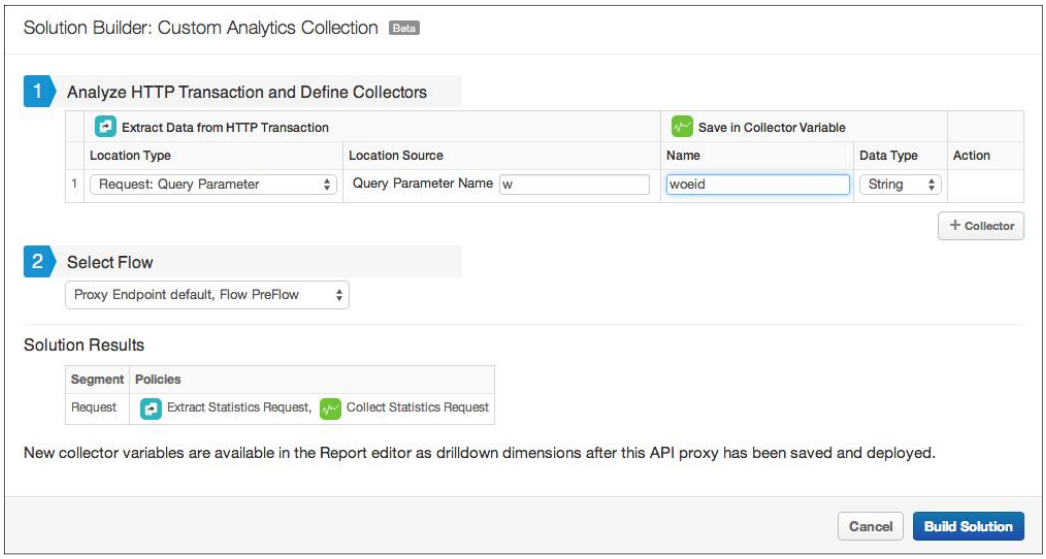
The Solution Builder dialog lets you configure analytics variables directly in the UI. This tool generates policies and attaches them to the API proxy for you. The policies extract variables of interest from requests or responses and pass the extracted variables to the SAP API Platform Analytics system.

> **Note**
>
> The Solution Builder creates new ExtractVariables and StatisticsCollector policies and gives them unique names. The Solution Builder does not let you go back and change these policies once they are created in a given proxy revision. If you wish to make changes, you can edit the generated policies directly the policy editor.

1. Go to the Overview page for your proxy in the SAP API Platform UI.
2. Choose Develop.
3. On the Develop page, select Custom Analytics Collection from the Tools menu. The Solution Builder dialog appears.
4. In the Solution Builder dialog, you first configure two policies: ExtractVariables and StatisticsCollector. Then, you configure where to attach those policies.
5. Specify the data you wish to extract:
   - Location Type: Select the type of data you wish to collect and where to collect it from. You can select data from the request or response side. For example, Request: Query Parameter or Response: XML Body.
   - Location Source: Identify the data you wish to collect. For example, the name of the query parameter or the XPath for XML data in the response body.
6. Specify a variable name (and type) that the StatisticsCollector policy will use to identify the extracted data. You can use any name. If you omit the name, the system selects a default for you.

   > **Note**
   >
   > The name you pick will appear in the Custom Dimensions dropdown menu in the Custom Report builder UI, as we'll see later in this section.
7. Pick where in the API proxy flow you wish to attach the generated policies ExtractVariables and StatisticsCollector. For guidance, see "Attaching policies to the ProxyEndpoint response Flow". To make things work properly, Policies must be attached to the API proxy Flow in the appropriate location. You need to attach the polices at a stage in the flow where the variables you are trapping are in scope (populated).
8. Choose +Collector to add more custom variables.
9. When you're done, choose Build Solution.
10. Save and deploy the proxy.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.



6.3.6 Use Custom Aggregations

This topic discusses how to create custom collections or "aggregates" of SAP API Platform analytics data.

6.3.6.1 About custom aggregation

SAP API Management Analytics Services automatically collects a set of pre-computed metrics. You can query this collected data to discover important information about how your API is performing. For example, the metric `message_count` keeps track of how many requests an API receives. Because this data is pre-aggregated (not mined on demand), it makes for fast processing time. To read more about computed metrics, see Analytics reference.

If you you have needs that exceed the automatically collected analytics data, you can also create custom aggregations. When you create a new custom aggregation, you can specify the metrics and dimensions that you wish to compute, and SAP API Platform collects and stores that data for you in the background, while your API is being used. Just as with the out-of-the-box pre-computed metrics, custom aggregations make access to the data much faster and more efficient than it would be to compute and return it on the fly.

You can use any of the four standard Analytics Services functions with any custom aggregate metrics. The functions `aresum, avg, max,` and `min`. For more information on metrics and functions, see Analytics reference.

6.3.6.2 Creating a new custom aggregation

SAP API Platform provides an API for creating custom aggregations of metric data. A custom aggregate can include up to three dimensions and five metrics, which you specify in the request payload. Here's an example of the POST command used to create a custom aggregate:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
$ curl -H "Content-type:application/json" -X POST -d \
'{
    "displayName": "my_custom_agg",
    “description” : “This is a sample custom aggregation”,
    “env”:”test”,
    "metrics": [
        "response_time",
        "message_count"
    ],
    "dimensions": [
        "is_error",
        "apiproxy",
        "developer_app"
    ],
}'  \
<Management-server-ip>:<port>/v1/o/stats/customaggregates \
-u myname:mypass
```

Notice that the request includes a list of metrics and dimensions. These dimensions and metrics are standard components that are used in SAP API Platform Analytics Services. For more information on dimensions and metrics, see "Dynamic Dimensions".

6.3.6.3 Starting and stopping aggregates

You can start and stop a custom aggregate's activity. When you stop an aggregation, the stored data up to that time is not deleted. After stopping an aggregation, you have up to 15 days to reactive it. After 15 days, the stored data is retained; however, you cannot restart the same aggregate. You need to create a new aggregate in that case.

When you activate or deactivate an aggregate, there is typically a time lag between when the command is issued and when it takes effect.

To stop a custom aggregate from collecting data, use this PUT call:

```
$ curl -H "Content-type:application/json" -X PUT -d \
'{
“status” : “inactive”
}'  \
<host>:<port>/v1/o/stats/customaggregates/{custom_agg_display_name} \
-u myname:mypass
```

To start an inactive aggregate, use this PUT call:

```
$ curl -H "Content-type:application/json" -X PUT -d \
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
'{
“status” : “active”
}'  \
http://<host:port>/v1/o/stats/customaggregates/{custom_agg_display_name} \
-u myname:mypass
```

6.3.6.4 Selecting dynamic dimensions from custom aggregates

Just as with other SAP API Platform statistical metrics, you can query custom aggregate metrics using the select query parameter. To drill into an aggregate data set, you simply need to provide the name of the custom aggregate as another query parameter, t, which specifies the unique name of the aggregate table stored in the database. For example, the following query returns the sum of response times stored in an aggregate with the unique name custom_agg_UUID.

```
$ curl -H "Content-type:application/json" -X GET -d \
<host>:<port>/v1/o/stats/apiproxy?select=sum(response_time)&t="custom_agg_UUID" \
-u myname:mypass
```

> **Note**
>
> Every aggregate has a unique name expressed by the value of the name attribute that is returned in the response when you create a new aggregate or retrieve an aggregate or list of aggregates through the API.The name is a UUID, a string of numbers and letters, which is assigned when the aggregate is created. In the following example, we simply use "custom_agg_UUID", but an acutal UUID would look something like this: my_custom_agg_250c1e623-e559-46b5-953e-bb532c245d3e.

For detailed information on using the select parameter to query SAP API Platform statistical dimensions, see "Dynamic Dimensions".

6.3.6.5 Filtering custom aggregate data sets with the query parameter

Just as with other SAP API Management Analytics, you can use the `filter` query parameter with custom aggregates. Filtering lets you "drill down" on a dimension. For example, to filter on API proxies that starts with "m":

```
$ curl -H "Content-type:application/json" -X GET -d \
<host>:<port>/v1/o/stats/apiproxy?select=sum(response_time)&t="custom_agg_UUID" \
&filter=(apiproxy LIKE 'm%') -u myname:mypass
```

For detailed information using the filter parameter, see "Filtering Results" and "Report Filters".

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.3.7 Troubleshooting with Analytics

6.3.7.1 Troubleshooting a Slow API

Data visualization is an important troubleshooting tool. SAP API Management is always collecting and analyzing data for your APIs, and visualization is the most powerful way to unlock, compare, contrast, and assess that data.

So, let's say you've been hearing from customers (perhaps via support calls, social media sites, or user forums) that one of your APIs is slow. Visualization can help. Where do you begin?

6.3.7.2 Consider and test for possible causes

Possible causes might include:

- If the slowness being seen by just one app or is it from multiple apps? If one app, then it might be a problem with the app.
- If it being seen by multiple users across multiple apps and users seem to be in the same geo location, then it could be a network issue
- If you're not seeing either of these issues, it could be an issue with SAP API Management. If you recently added or updated a policy. It could be configured incorrectly.
- If the total response time is being reported as high, but the average endpoint response time has not changed then it might be an SAP API Management issue. If the average endpoint response time is also high then it could be an issue in the network between SAP API Management and the target server, or an internal application server.

6.3.7.3 Visualize which API is slow

To discover which API is underperforming, one approach is to compare the average response times of all your APIs to see if one of them is out of line.

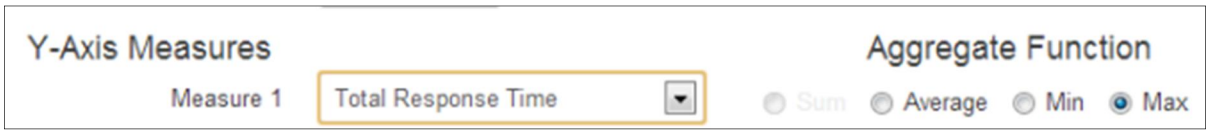
1. Select API Proxies from the APIs menu.
2. In the Performance section of the API Proxies page, select Average Response Time from the Performance menu.
3. Select a range of dates to evaluate.
4. Select which APIs you wish to evaluate.
5. In the plot, look for sudden spikes or gradual increases in response time.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

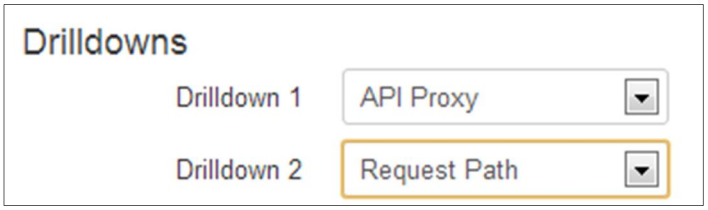
6.3.7.4 Visualize with a custom report

If you suspect which API is causing a problem, you can create a custom report that includes charts and plots comparing specific metrics that you select. For example, you can visualize the maximum latency time by app and resource.

1. Select Custom Reports from the Analytics menu.
2. In the Custom Reports page, choose + Custom Report.
3. Fill out the Custom Report basics, select your sampling rate, and the environment you wish to test.
4. In the Y-Axis Measures section, select Total Response Time and Max for the Aggregate Function:



5. Specify Drilldowns for the API Proxy and Request Path:



6.3.7.5 Analyze the resulting report

The new latency report will show you the response times by API and then by each resource within an API. By combining this information with what you know about your network architecture, you can quickly find issues that may be related to your infrastructure.

6.3.7.6 Drill down

You can drill down on the worst performing API (the one with the highest latency).

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6.3.7.7 Act

Now that you know which resource is performing badly, you can examine your network to see if there's a service issue, or you can add a 3rd dimension like Developer App to see which apps are impacted by this slow resource, or developer to figure out which developers are impacted by the slow resource.

As you can see, SAP API Platform Analytics Services includes multiple ways to approach a specific problem, and visualization plays an important role in any troubleshooting process.

6.3.7.8 What does an analytics entity value "(not set)" mean?

As you review your analytics, you may see an entity value of (not set) displayed, including the parenthesis, for your API Proxies, Product, Developer, and Developer Apps dimensions. This may or may not be an issue.

See below for possible explanations of why you are seeing a (not set) entity.

- You see a (not set) entity for the API Proxies dimension.

  This means that a call to an API proxy reached the SAP API Management routers, but was malformed, never reached the proxy, and the message processor returned a 404 status code with a classification error. This means that there are some callers that are making malformed calls.
- You see an (not set) entity for the Product dimension entity.

  This means that not all of your API proxies and developer apps are using products.
- You see an (not set) entity for the Developer dimension.

  This means that some of your traffic is being generated by unregistered developers. This traffic may originate with an internal-use or public API.
- You see a (not set) entity for the Developer Apps dimension.

  This means that some of your traffic is being generated by unregistered apps. This traffic may originate with an internal-use or public API.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7 References

7.1 Analytics Command Reference

This section provides a list a preconfigured URIs for specific applications of SAP API Management Analytics Services. The requests below assume the environment is 'prod'. If you require analytics on another environment (such as 'test'), `modify /environments/prod to/environments/test`.

7.1.1 API

Requests by API

```
$ curl -u myname:mypass
https://<host:port>/v1/o/{myorg}/environments/prod/stats/apis?select=sum(message_co
unt)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Total Response Time by API

```
$ curl -u myname:mypass
https://<host:port>/v1/o/{myorg}/environments/prod/stats/apis?select=avg(total_resp
onse_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Total Response Time by API

```
$ curl -u myname:mypass
https://<host:port>/v1/o/{myorg}/environments/prod/stats/apis?select=max(total_resp
onse_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Total Response Time by API

```
$ curl -u myname:mypass
https://<host:port>/v1/o/{myorg}/environments/prod/stats/apis?select=min(total_resp
onse_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Target Response Time by API

```
$ curl -u myname:mypass
https://<host:port>/v1/o/{myorg}/environments/prod/stats/apis?select=avg(target_res
ponse_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Target Response Time by API

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=max(target_response_
time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Target Response Time by API

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=min(target_response_
time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Errors by API

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=count(is_error)&time
Range=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Request Size by API

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=max(request_size)&ti
meRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Request Size by API

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=min(request_size)&ti
meRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg Request Size by API

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=avg(request_size)&ti
meRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Response Size by API

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=max(response_size)&t
imeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Response Size by API

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=min(response_size)&t
imeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg Response Size by API

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=avg(response_size)&t
imeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Response Processing Latency by API

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=avg(response_process
ing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Response Processing Latency by API

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=max(response_process
ing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Response Processing Latency by API

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=min(response_process
ing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Request Processing Latency by API

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=avg(request_processi
ng_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Request Processing Latency by API

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=max(request_processi
ng_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Request Processing Latency by API

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=min(request_processi
ng_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

7.1.2 Developers

App count by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=sum(app_count)&timeR
ange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

End User count by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=sum(user_count)&time
Range=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Requests by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=sum(message_count)&t
imeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Total Response Time by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=avg(total_response_t
ime)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Total Response Time by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=max(total_response_t
ime)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Total Response Time by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=min(total_response_t
ime)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Target Response Time by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=avg(target_response_
time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Target Response Time by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=max(target_response_
time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Min Target Response Time by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=min(target_response_
time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Errors by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=count(is_error)&time
Range=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Request Size by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=max(request_size)&ti
meRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Request Size by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=min(request_size)&ti
meRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg Request Size by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=avg(request_size)&ti
meRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Response Size by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=max(response_size)&t
imeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Response Size by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=min(response_size)&t
imeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg Response Size by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=avg(response_size)&t
imeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Response Processing Latency by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=avg(response_process
ing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Response Processing Latency by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=max(response_process
ing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Response Processing Latency by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=min(response_process
ing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Avg. Request Processing Latency by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=avg(request_processi
ng_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Request Processing Latency by Developers

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/devs?select=min(request_processi
ng_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

7.1.3 Apps

User count by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=sum(user_count)&time
Range=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Requests by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=sum(message_count)&t
imeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Total Response Time by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=avg(total_response_t
ime)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Total Response Time by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=max(total_response_t
ime)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Total Response Time by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=min(total_response_t
ime)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Target Response Time by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=avg(target_response_
time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Target Response Time by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=max(target_response_
time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Target Response Time by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=min(target_response_
time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Errors by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=count(is_error)&time
Range=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Request Size by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=max(request_size)&ti
meRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Request Size by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=min(request_size)&ti
meRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg Request Size by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=avg(request_size)&ti
meRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Response Size by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=max(response_size)&t
imeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Response Size by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=min(response_size)&t
imeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg Response Size by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=avg(response_size)&t
imeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Response Processing Latency by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=avg(response_process
ing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Response Processing Latency by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=max(response_process
ing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Response Processing Latency by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=min(response_process
ing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Request Processing Latency by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=avg(request_processi
ng_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Request Processing Latency by Apps

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=max(request_processi
ng_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Request Processing Latency by Apps

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apps?select=min(request_processi
ng_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

7.1.4 API Products

App count by API Products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apiproducts?select=sum(app_count
)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Developer count by API Products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apiproducts?select=sum(developer
_count)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Requests by Products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=sum(message_c
ount)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Total Response Time API products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=avg(total_res
ponse_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Total Response Time API products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=max(total_res
ponse_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Total Response Time API products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=min(total_res
ponse_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Target Response Time API products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=avg(target_re
sponse_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Target Response Time API products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=max(target_re
sponse_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Min Target Response Time API products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=min(target_response_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Errors by API Products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=count(is_error)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Request Size by API Products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=max(request_size)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Request Size by API Products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=min(request_size)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg Request Size by API Products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=avg(request_size)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Response Size by API Products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=max(response_size)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

MIn Response Size by API Products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=min(response_size)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg Response Size by API Products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=avg(response_size)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Response Processing Latency by API products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=avg(response_processing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Response Processing Latency by API products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=max(response_processing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Response Processing Latency by API products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=min(response_processing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Avg. Request Processing Latency by API products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=avg(request_processing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Request Processing Latency by API products

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/api_product?select=min(request_processing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

7.1.5 API Resources

Requests by API Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/request_path?select=sum(message_count)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Total Response Time by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/request_path?select=avg(total_response_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Total Response Time by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=max(total_response_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Total Response Time by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=min(total_response_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Target Response Time by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/request_path?select=avg(target_response_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Target Response Time by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=max(target_response_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Target Response Time by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=min(target_response_time)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Errors by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/request_path?select=count(is_error)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Max Request Size by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/request_path?select=max(request_
size)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Request Size by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/request_path?select=min(request_
size)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg Request Size by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/request_path?select=avg(request_
size)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Response Size by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/request_path?select=max(response
_size)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Response Size by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/request_path?select=min(response
_size)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg Response Size by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/request_path?select=avg(response
_size)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Response Processing Latency by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/request_path?select=avg(response
_processing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Response Processing Latency by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=max(response_process
ing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Response Processing Latency by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=min(response_process
ing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Request Processing Latency by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/request_path?select=avg(request_
processing_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Request Processing Latency by API-Resource

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=max(request_processi
ng_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Request Processing Latency by API-Resource

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/apis?select=min(request_processi
ng_latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

7.1.6 API program-wide analytics

Requests over Time

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats?select=sum(message_count)&timeRa
nge=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Average Total Response Time

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=avg(total_response_time)
&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Total Response Time

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=max(total_response_time)
&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Total Response Time

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=min(total_response_time)
&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Average Target Response Time

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=avg(target_response_time
)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Target Response Time

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=max(target_response_time
)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Target Response Time

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=min(target_response_time
)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Errors over Time

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=count(is_error)&timeRang
e=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Request Size over Time

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=max(request_size)&timeRa
nge=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Min Request Size over Time

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=min(request_size)&timeRa
nge=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg Request Size over Time

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=avg(request_size)&timeRa
nge=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Response Size over Time

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=max(response_size)&timeR
ange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Response Size over Time

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=min(response_size)&timeR
ange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg Response Size over Time

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=avg(response_size)&timeR
ange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Avg. Response Processing Latency

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=avg(response_processing_
latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Response Processing Latency

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=max(response_processing_
latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Response Processing Latency

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=min(response_processing_
latency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Average Request Processing Latency

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=avg(request_processing_l
atency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Max Request Processing Latency

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=max(request_processing_l
atency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```

Min Request Processing Latency

```
$ curl -u myname:mypass
<host>:<port>/v1/o/{myorg}/environments/prod/stats/?select=min(request_processing_l
atency)&timeRange=9/24/2012%2000:00~10/25/2012%2000:00&timeUnit=day
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.2 Analytics Dimensions and Metrics

This topic is a reference. To learn how to use the dimensions and metrics that are detailed in this reference, start with the Analytics Services overview.

Analytics services exposes APIs for three types of dimensions:

- Dedicated dimensions: Analytics that are computed in aggregate based on pre-defined dimensions and metrics, with no detailed drill-down available. These are used primarily to populate default dashboards and operational performance graphs in the management UI.
- Dynamic dimensions: Analytics for a pre-defined set of dimensions that are computed only when you build custom reports. As is explained below, these require you to specify a function, for example, the average, minimum value, or maximum value, to calculate metrics against a dimension, such as the response time of an API proxy.
- Custom dimensions: Analytics that you use to build reports for data that is specific to your business, and which is not measured by the Analytics above. Custom Analytics require the definition and population of custom variable as well as requiring a function to be defined. For instructions on building reports for custom dimensions, see Analyze API message content using custom analytics.
- Virtual dimensions: The Analytics system generates virtual dimensions from the user-agent, timestamp, and IP associated with each API call. Virtual dimensions are provided out of the box.

7.2.1 Dedicated dimensions

Dedicated dimensions are associated with aggregate, precomputed metrics. For speed and scale, Analytics Services automatically calculates these metrics. This makes them very simple to use, but limits their flexibility. For example, functions are not supported against dedicated dimensions because the dimensions have been pre-aggregated; they have 'built-in' functions. For this reason, dedicated dimensions do not appear as available options in the Management UI during report creation. Rather, dedicated dimensions are used to generate operational dashboards in the Management UI. You can access the same dedicated dimensions to build your own dashboards using the Analytics API.

The 'select' query parameter for dedicated dimensions accepts a metric name but does not support a function (since the function is already specified for the associated pre-computed metric).

Name	Description	Computed Metrics
`/stats/`	Total traffic across all API proxies in an environment	message_count
`/stats/apis`	Metrics across all API proxies	message_count, error_count, total_response_time, max_response_time, min_response_time,

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Computed Metrics
	in a specific environment.	data_exchange_size, end_point_response_time, tps, tpm, tph
`/stats/apiproducts`	Metrics across all API products in an organization	message_count, error_count, total_response_time, max_response_time, min_response_time, data_exchange_size, end_point_response_time
`/stats/apps`	Metrics across all apps in an organization	message_count, error_count, user_count total_response_time, max_response_time, min_response_time, end_point_response_time, tps, tpm, tph
`/stats/devs`	Metrics across all developers in an organization	message_count, error_count, app_count, user_count, tps, tpm, tph
`/stats/ax_geo_country`	Metrics across all countries	{function}(message_count), {function}(total_response_time)

7.2.2 Computed metrics

Computed metrics are pre-defined (metric + function) calculations that are used to generate reports for dedicated dimensions. Computed metrics enable the system to provide operational analytics in a way that is highly optimized. Computed metrics are used to populate dashboards in the API management UI.

Name	Description	Method of calculation
`message_count`	Total number of request messages received by all API proxies	message_count
`error_count`	Total number of all error messages (request, response) across all API proxies	message_count
`app_count`	Total number of all apps provisioned	message_count
`api_count`	Total number of API proxies	message_count
`total_response_time`	Total traffic across all API proxies in an environment	message_count
`max_response_time`	The highest value, in milliseconds, for a complete roundtrip transaction on SAP API Management, including network	message_count

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Method of calculation
	latency and processing time by the backend (target) service	
`min_response_time`	The smallest value, in milliseconds, for a complete roundtrip transaction on SAP API Management, including network latency and processing time by the backend (target) service	message_count
`data_exchange_size`	The size, in kilobytes, of the inbound request message plus the size, in kilobytes, of the outbound response message	message_count
`end_point_response_time`	The time, in milliseconds, between the TargetEndpoint response is sent and the request is received by the TargetEndpoint	message_count
`tps`	Number of transactions per second. If tps is specified, the timeUnit query parameter must be greater than one second.	message_count
`tpm`	Number of transactions per minute. If tpm is specified, the timeUnit query parameter must be greater than one minute.	message_count
`tph`	Number of transactions per hour. If tph is specified, the timeUnit query parameter must be greater than one hour.	message_count

7.2.3 Metrics

Name	Description	Functions
cache_hit	A response served from the Cache on SAP API Management (and which therefore did not result in a request being forwarded to the backend service)	Sum, Avg
is_error	Any transaction that results in a transport or application error.	Sum, Avg

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Functions
request_processing_latency	Duration of processing, measured in milliseconds, of the request pipeline, encompassing ProxyEndpoint request and TargetEndpoint request segments.	Avg, Min, Max
request_size	The size, measured in kilobytes, of a request message received by a ProxyEndpoint from the client app.	Sum, Avg, Min, Max
response_processing_latency	Duration of processing, measured in milliseconds, of the response pipeline, encompassing ProxyEndpoint response and TargetEndpoint response segments.	Avg, Min, Max
response_size	The size, measured in kilobytes, of a response message sent from a ProxyEndpoint to a client app.	Sum, Avg, Min, Max
target_response_size	The size, measured in kilobytes, of a response message received by a TargetEndpoint from the backend service.	Avg, Min, Max
total_response_time	Duration of processing, measured in milliseconds, of a request message received by a ProxyEndpoint from a client, until the request message is returned by the ProxyEndpoint, encompassing both request and response Flows plus backend processing time. In other words, total processing time by an API proxy minus the network latency between client app and SAP API Management.	Avg, Min, Max
message_count	Total number of request messages processed by the API proxy.	Sum, Avg, Min, Max

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.2.4 Functions

Name	Description	Metrics
sum	An operation that adds all values in the database for a given metric, resulting in the total of all values.	cache_hit, is_error, request_size, response_size, target_response_size, total_response_time, message_count
avg	An operation that adds all values in the database for a given metric, and then divides by the number of entries in the data set, resulting in a single value that characterizes the data set as a whole.	cache_hit, is_error, request_processing_latency, request_size, response_processing_latency, response_size, message_count
min	The lowest numerical value in a given set of metrics.	request_processing_latency, request_size, response_processing_latency, response_size, message_count
max	The highest numerical value in a given set of metrics.	request_processing_latency, request_size, response_processing_latency, response_size, message_count

7.2.5 Dynamic Dimensions

Dynamic dimensions are a set of pre-defined dimensions that are not associated with precomputed metrics. The metrics are computed at the time that you make a request for a report. This provides you with more flexibility to slice-and-dice the measurements that are recorded by Analytics services.

Dynamic dimensions enable you to specify a function and a raw metric in the select parameter. Dynamic dimensions are used to generate custom reports.

Example

```
select=sum(message_count)
or
select=avg(total_response_time)
```

Note

Dynamic dimensions should not be confused with custom dimensions. Custom dimensions are derived business-specific message content that is unique to a particular API. Custom dimensions are defined

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

using the StatisticsCollector policy policy (see Capture analytics from API traffic, or see Analyze API message content using custom analytics for a full example).

Each dynamic dimension supports a set of metrics and functions. (When using the Management UI, only the functions available for a given metric can be selected. Radio buttons for unavailable functions are disabled.)

Dynamic dimensions enable you to 'slice and dice' the metric/function combinations..

For example:

Each dynamic dimension supports a set of metrics and functions. (When using the Management UI, only the functions available for a given metric can be selected. Radio buttons for unavailable functions are disabled.)

Dynamic dimensions enable you to 'slice and dice' the metric/function combinations..

For example:

```
<host>:<port>/v1/o/{org_name}/environments/{env_name}/stats/access_token?select={function}({metric})&timeRange={startTime~endTime}&timeUnit
```

```
<host>:<port>/v1/o/{org_name}/environments/{env_name}/stats/client_id?select={function}({metric})&timeRange={startTime~endTime}&timeUnit
```

.

```
<host>:<port>/v1/o/{org_name}/environments/{env_name}/stats/client_ip?select={function}({metric})&timeRange={startTime~endTime}&timeUnit
```

Dynamic dimensions

Name	Description	Metrics	Functions
`/stats/access_token`	App end user's OAuth access token. (OAuth policies must be enforced for this dimension to be populated.)	metrics	functions
`/stats/api_product`	An API product in an organization.	metrics	functions
`/stats/apiproxy`	An API proxy	metrics	functions
`/stats/apiproxy_revision`	A specific revision of an API proxy	metrics	functions
`/stats/client_host`	The network name of the computer hosting the client app	metrics	functions
`/stats/client_id`	The API key, also known as consumer key, of the client app.	metrics	functions
`/stats/client_ip`	The IP address of the client app.	metrics	functions

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Metrics	Functions
`/stats/developer`	The developer associated with the client app, identified by internal developer ID (generated by SAP API Management when a developer registers in an organization).	metrics	functions
`/stats/developer_app`	The name of the developer app submitting the request. (SAP API Management groups apps by developer.)	metrics	functions
`/stats/developer_email`	The email address of the developer.	metrics	functions
`/stats/environment`	The environment in an organization in which an API proxy is deployed.	metrics	functions
`/stats/flow_resource`	The URI (defined as an API resource by a conditional Flow)	metrics	functions
`/stats/gateway_flow_id`	The name of the API Proxy Flow	metrics	functions
`/stats/organization`	The organization where metrics are gathered.	metrics	functions
`/stats/proxy`	The name of the ProxyEndpoint	metrics	functions
`/stats/proxy_basepath`	The URI path fragment defined as the base path for an API proxy in the ProxyEndpoint configuration.	metrics	functions
`/stats/proxy_client_ip`	The IP address of the ProxyEndpoint.	metrics	functions
`/stats/proxy_pathsuffix`	The URI fragment that follows the proxy base path.	metrics	functions
`/stats/request_path`	The full URI path of the request message.	metrics	functions

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Metrics	Functions
`/stats/request_uri`	The full URI of the request message	metrics	functions
`/stats/request_verb`	The HTTP verb associated with the request message	metrics	functions
`/stats/target`	The name of the TargetEndpoint	metrics	functions
`/stats/target_host`	The name of the network host where the TargetEndpoint is deployed.	metrics	functions
`/stats/target_basepath`		metrics	functions
`/stats/target_ip`	The IP address of the TargetEndpoint	metrics	functions
`/stats/target_url`	The complete URL of the target service (that is, the backend service)	metrics	Functions
`/stats/virtual_host`	The name of the VirtualHost (that is, the named network address configuration) invoked by the client app	metrics	functions

7.2.6 Virtual dimensions

The Analytics system generates virtual dimensions from the user-agent, timestamp, and IP associated with each API call. Virtual dimensions are provided out of the box. You cannot create new virtual dimensions. The virtual dimensions include:

Name	Description
OS version	user-agent string
OS family	user-agent string
Device family	user-agent string
Time of day	timestamp
Day of week	timestamp
City	IP address
Country	IP address

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description
Timezone	IP address

7.2.7 Report Filters

A filter is expressed as a query parameter in an API request to build a report. Filters can be applied to any API request.

Filter on API proxies with the name books or music:

```
filter=(apiproxy IN 'books','music')
```

Filter on API proxies with names that start with "m":

```
filter=(apiproxy LIKE 'm%')
```

Filter on response codes that do not start with "m":

```
filter=(apiproxy NOT LIKE 'm%')
```

Filter on response codes between 400 and 599:

```
filter=(responsecode IN '4xx','5xx')
```

Filter on response code 500:

```
filter=(responsecode EQ 500)
```

Filter on metrics for non-error messages:

```
filter=(is_error EQ 0)
```

Token	Description
`in`	Include in list
`notin`	Exclude from list
`eq`	Equals, ==
`ne`	Not equal to, !=
`nt`	Greater than, >
`lt`	Less than, <
`ge`	Greater than or equal to, >=
`le`	Less than or equal to, <=
`like`	Returns true if the string pattern matches the supplied pattern.
`not like`	Returns false if the string pattern matches the supplied pattern.
`similar to`	Returns true or false depending on whether its pattern matches the given string. It is similar to LIKE, except that it interprets the pattern using the SQL standard's definition of a regular expression.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Token	Description
`not similar to`	Returns false or true depending on whether its pattern matches the given string. It is similar to NOT LIKE, except that it interprets the pattern using the SQL standard's definition of a regular expression.
`and`	Lets you use 'and' logic to include more than one filter expression. The filter includes data that meets all the conditions.
`Or`	Lets you use 'or' logic to evaluate different possible filter expressions. The filter includes data that meets at least one of the conditions.

7.3 API Proxy Configuration Reference

As a developer working with the SAP API Management, your primary development activities involve configuring API proxies that function as proxies for APIs or backend services. This section is a reference of all configuration elements available to you when building API proxies.

If you are learning how to build API proxies, it is recommended that you begin with the topic Build a simple API proxy.

7.3.1 API proxy structure

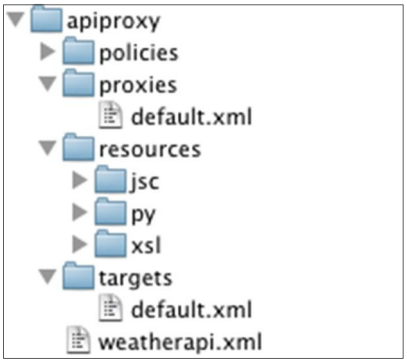
An API proxy consists of the following configuration:

Base Configuration	Primary configuration settings for an API proxy.
ProxyEndpoint Configuration	Settings for the inbound HTTP connection (from requesting apps to SAP API Management), request and response flows, and policy attachments. See ProxyEndpoint.
TargetEndpoint Configuration	Settings for the outbound HTTP connection (from SAP API Management to the backend service), request and response flows, and policy attachments. See TargetEndpoint.
Flows	ProxyEndpoint and TargetEndpoint request and response pipelines to which polices can be attached. See Flows.
Policies	XML-formatted configuration files that conform to the SAP API Management policy schemas. See Policies.
Resources	Scripts, JAR files, and XSLT files referenced by policies to execute custom logic. See Resources.
Base Configuration	Primary configuration settings for an API proxy. See Base Configuration.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

API proxy directory structure and contents

The components in the table above are defined by configuration files in the following directory structure:



7.3.2 Configuration files and directory structure of an API proxy

This section explains the configuration files and directory structure of an API proxy.

- Base Configuration.
- ProxyEndpoint
- TargetEndpoint.
- Advanced TargetEndpoint Configuration
- Policies
- Flows
- Resources

7.3.3 Base Configuration

`/apiproxy/weatherapi.xml`

The base configuration for an API proxy, which defines the name of the API proxy. The name must be unique within an organization.

Sample configuration:

```
<APIProxy name="weatherapi">
</APIProxy>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

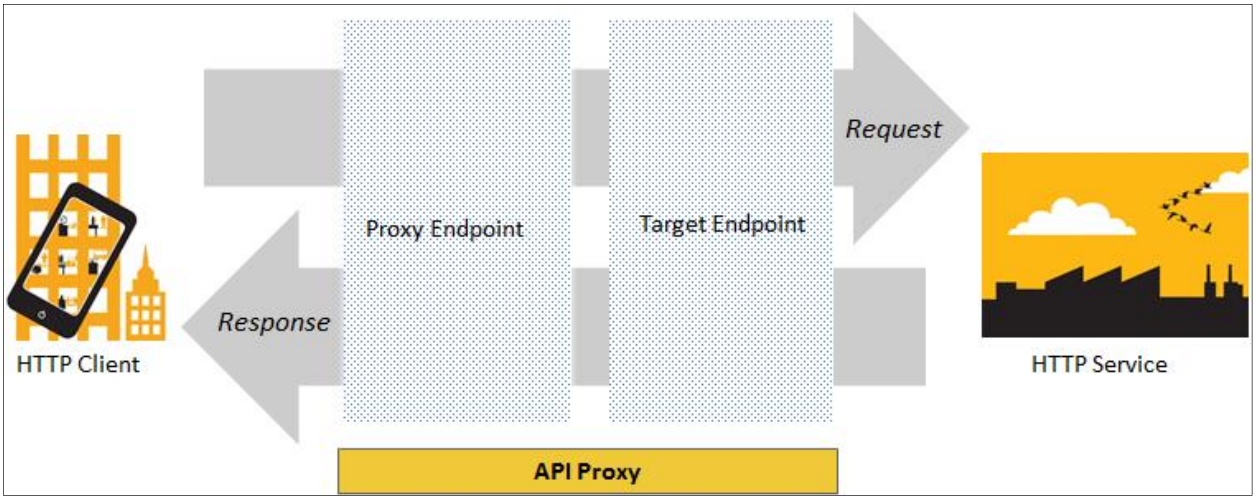
Base Configuration Elements

Name	Description	Default	Required?
API Proxy			
name	The name of the API proxy, which must be unique within an organization. The characters you are allowed to use in the name are restricted to the following: A-Z0-9._\-$ %.	N/A	Yes
revision	The revision number of the API proxy configuration. You do not need to explicitly set the revision number, since SAP API Management automatically tracks the current revision of the API proxy.	N/A	No
ConfigurationVersion	The version of the API proxy configuration schema to which this API proxy conforms. The only supported value currently is majorVersion 4 and minorVersion 0. This setting may be used the in the future to enable evolution of the APi proxy format.	4.0	No
Description	A textual description of the API proxy. If provided, the description will display in the SAP API Platform management UI.	N/A	No
DisplayName	A user-friendly name that may be different from the name attribute of the API proxy configuration.	N/A	No
Policies	A list of policies in the /policies directory of this API proxy. You will normally only see this element when the API proxy was created using the SAP API Platform management UI. This is simply a 'manifest' setting, designed to provide visibility into the contents of the API proxy.	N/A	No
ProxyEndpoints	A list of ProxyEndpoints in the /proxies directory of this API proxy. You will normally only see this element when the API proxy was created using the SAP API Platform management UI. This is simply a 'manifest' setting, designed to provide visibility into the contents of the API proxy.	N/A	No
Resources	A list of resources (JavaScript, Python, XSLT) in the /resources directory of this API proxy. You will normally only see this element when the API proxy was created using the SAP API Platform management UI. This is simply a 'manifest' setting, designed to provide visibility into the contents of the API proxy.	N/A	No
TargetServers	A list of TargetServers referenced in any TargetEndpoints of this API proxy. You will normally only see this element when the API proxy was created using the SAP API Platform management UI. This is simply a 'manifest' setting, designed to provide visibility into the contents of the API proxy.	N/A	No

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Default	Required?
TargetEndpoints	A list of TargetEndpoints in the `/targets` directory of this API proxy. You will normally only see this element when the API proxy was created using the SAP API Platform management UI. This is simply a 'manifest' setting, designed to provide visibility into the contents of the API proxy.	N/A	No

7.3.4 ProxyEndpoint



`/apiproxy/proxies/default.xml`

The ProxyEndpoint configuration defines the inbound (client-facing) interface for an API proxy. When you configure a ProxyEndpoint, you are setting up a network configuration that defines how client applications ('apps') should invoke the proxied API.

The following sample ProxyEndpoint configuration would be stored under `/apiproxy/proxies`:

```
<ProxyEndpoint name="default">
  <PreFlow/>
  <Flows/>
  <PostFlow/>
  <HTTPProxyConnection>
    <BasePath>/weather</BasePath>
    <VirtualHost>default</VirtualHost>
  </HTTPProxyConnection>
  <FaultRules/>
  <DefaultFaultRule/>
  <RouteRule name="default">
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <TargetEndpoint>default</TargetEndpoint>
  </RouteRule>
</ProxyEndpoint>
```

7.3.5 ProxyEndpoint Configuration Elements

The required configuration elements in a basic ProxyEndpoint are:

ProxyEndpoint			
`name`	The name of the ProxyEndpoint. Must be unique within the API proxy configuration, when (in rare cases) multiple ProxyEndpoints are defined. The characters you are allowed to use in the name are restricted to the following: A-Z0-9._\-$ %.	N/A	Yes
`PreFlow`	Defines the policies in the PreFlow flow of a request or response.	N/A	Yes
`Flows`	Defines the policies in the conditional flows of a request or response.	N/A	Yes
`PostFlow`	Defines the policies in the PostFlow flow of a request or response.	N/A	Yes
HTTPProxyConnection	Defines the network address and URI path associated with the API proxy		
`BasePath`	A required string that uniquely identifies the URI path used by SAP API Management to route incoming messages to the proper API proxy. The BasePath is a URI fragment (for example/weather) appended to the base URL for an environment (for example, `http://apifactory.net) by a requesting client`. BasePath must be unique within an environment. Uniqueness is validated by API Platform when an API proxy is generated or imported. **Using a wildcard in base paths** You can use one or more `/*/` wildcards in API proxy base paths to future-proof your proxies. For example, a base path of `/team/*/members` allows clients to call **`https://[host]/team/blue/members andhttps://[host]/team/green/members`** without you needing to create new API proxies to support new teams. Note that `/**/` is not supported.	/	Yes

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

`VirtualHost`	Associates an API proxy with specific base URLs for an environment. A VirtualHost is a named configuration that defines one or more URLs for an environment. The named VirtualHosts defined for a ProxyEndpoint determine the domains and ports on which an API proxy is exposed, and, by extension, the URL that apps use to invoke an API proxy. By default, two named VirtualHosts are defined for an environment: `default` and `secure`. An organization may also define custom domains. To ensure that an API proxy is available only over HTTPs, for example, set the VirtualHost in the HTTPProxyConnection to secure.	default	Yes
`Properties`	A set of optional HTTP configuration settings can be defined as properties of a ProxyEndpoint. See Endpoint properties reference.	N/A	No
`FaultRules`	Defines how the ProxyEndpoint reacts to an error. A fault rule specifies two items: · A Condition that specifies the fault to be handled based on the pre-defined category, subcategory, or name of the fault · One or more policies that define the behavior of the fault rule for the corresponding Condition	N/A	No
`DefaultFaultRule`	Handles any errors (system, transport, messaging or policy) that are not explicitly handled by another FaultRule. See Fault handling.	N/A	No
`RouteRule`	Defines the destination of inbound request messages after processing by the ProxyEndpoint request pipeline. Usually, the RouteRule points to a named TargetEndpoint configuration, but it can also point directly to a URL.		
`Name`	Required attribute, which provides a name for the RouteRule. The characters you are allowed to use in the name are restricted to the following: A-Z0-9._\-$%. For example, `Cat2 %_` is a legal name.	N/A	Yes
`Condition`	An optional conditional statement used for dynamic routing at runtime. Conditional RouteRules are useful, for example, to enable content-based routing to support backend versioning.	N/A	No
`TargetEndpoint`	An optional string that identifies a named TargetEndpoint configuration. A named TargetEndpoint is any TargetEndpoint defined in the same API proxy under the `/targets` directory). By naming a TargetEndpoint, you indicate where request messages should be forwarded after processing by the	N/A	No

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

	ProxyEndpoint request pipeline. Note that this is an optional setting. A ProxyEndpoint may call a URL directly. For example, a JavaScript or functioning in the role of an HTTP client, may perform the basic duty of a TargetEndpoint, which is to forward requests to a backend service.		
URL	An optional string that defines an outbound network address called by the ProxyEndpoint, bypassing any TargetEndpoint configurations that might be stored under /targets	N/A	No

7.3.6 How to Configure RouteRules

A named TargetEndpoint refers to a configuration file under `/apiproxy/targets` to which the RouteRule forwards a request after processing by the ProxyEndpoint.

For example, the following RouteRule refers to the configuration /apiproxy/targets/myTarget.xml:

```
<RouteRule name="default">
  <TargetEndpoint>myTarget</TargetEndpoint>
</RouteRule>
```

7.3.7 Conditional Routes

RouteRules can be chained to support dynamic routing at runtime. Inbound requests can be routed to named TargetEndpoint configurations, directly to URLs, or to a combination of the two, based on HTTP headers, message content, query parameters, or contextual information such time of day, locale, etc.

Conditional RouteRules work like other conditional statements on SAP API Management. See Conditions reference and Variables reference.

For example, the following RouteRule combination first evaluates the inbound request to verify the value of an HTTP header. If the HTTP header routeTo has the value TargetEndpoint1, then the request is forwarded to the TargetEndpoint named TargetEndpoint1. If not, then the inbound request is forwarded to `http://api.mycompany.com/v2`.

```
<RouteRule name="MyRoute">
  <Condition>request.header.routeTo = "TargetEndpoint1"</Condition>
  <TargetEndpoint>TargetEndpoint1</TargetEndpoint>
</RouteRule>
<RouteRule name="default">
  <URL>http://api.mycompany.com/v2</URL>
</RouteRule>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.3.8 Null Routes

A null RouteRule can be defined to support scenarios in which the request message does not need to be forwarded to the TargetEndpoint. This is useful when the ProxyEndpoint performs all of the necessary processing, for example by using JavaScript to call an external service or retrieving data from a lookup to the API Platform' key/value store.

For example, the following defines a null Route:

```
<RouteRule name="GoNowhere"/>
```

Conditional null Routes can be useful. In the following example, a null Route is configured to execute when an HTTP header request.header.X-DoNothing have a value other than null.

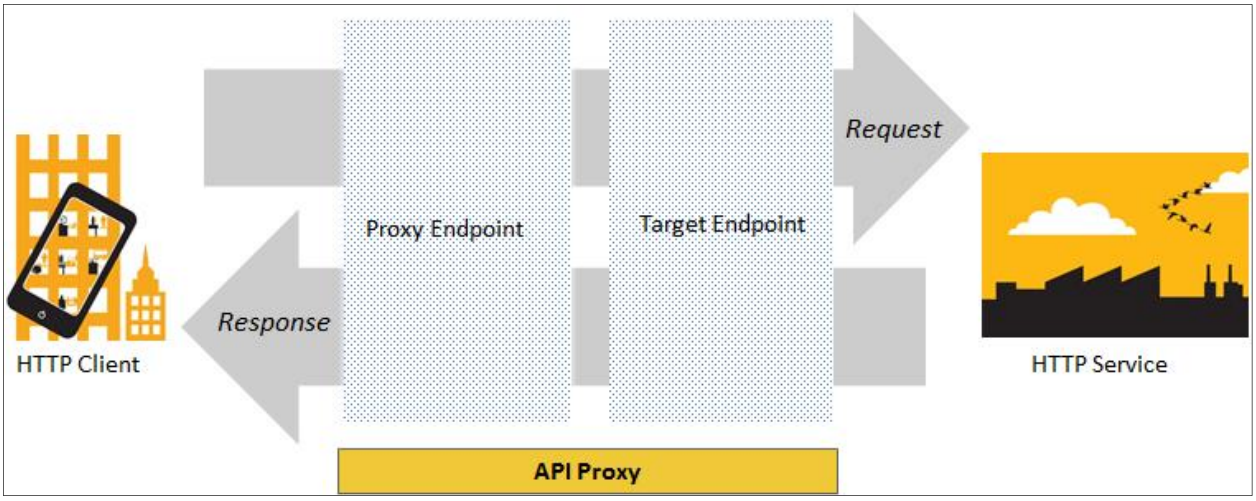
```
<RouteRule name="DoNothingOnDemand">
  <Condition>request.header.X-DoNothing != null</Condition>
</RouteRule>
```

Remember, RouteRules can be chained, so a conditional null Route would typically be one component of a set of RouteRules designed to support conditional routing.

A practical use of a conditional null Route would be in support of caching. By using the value of the variable that is set by the Cache policy, you can configure an API proxy to execute the null Route when an entry is served from the cache.

```
<RouteRule name="DoNothingUnlessTheCacheIsStale">
  <Condition>lookupcache.LookupCache-1.cachehit is true</Condition>
</RouteRule>
```

7.3.9 TargetEndpoint



TargetEndpoint is the outbound equivalent of the ProxyEndpoint. A TargetEndpoint functions as an HTTP client to a backend service or API--it sends requests and receives responses. An API proxy can contain zero or more TargetEndpoints. (ProxyEndpoints can be configured to call URLs directly --an API proxy with no TargetEndpoints

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

usually contains a ProxyEndpoint that either directly calls a backend service, or that is configured to call a service using JavaScript.)

7.3.10 TargetEndpoint Configuration

/targets/default.xml

The TargetEndpoint defines the outbound connection from SAP API Management to a backend service.

Sample TargetEndpoint configuration:

```
<TargetEndpoint name="default">
  <PreFlow/>
  <Flows/>
  <PostFlow/>
  <HTTPTargetConnection>
    <URL> (Back-end URL) </URL>
  </HTTPTargetConnection>
  <FaultRules/>
  <DefaultFaultRule/>
  <ScriptTarget/>
  <LocalTargetConnection/>
</TargetEndpoint>
```

7.3.11 TargetEndpoint Configuration Elements

Name	Description	Default	Required?
TargetEndpoint			
`name`	The name of the TargetEndpoint, which must be unique within the API proxy configuration. The name of the TargetPoint is used in the ProxyEndpoint RouteRule to direct requests for outbound processing. The characters you are allowed to use in the name are restricted to the following: `A-Z0-9._\-$ %.`	N/A	Yes
`PreFlow`	Defines the policies in the PreFlow flow of a request or response.	N/A	Yes
`Flows`	Defines the policies in the conditional flows of a request or response.	N/A	Yes
`PostFlow`	Defines the policies in the PostFlow flow of a request or response.	N/A	Yes
HTTPTargetConnection			

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Default	Required?
`URL`	Defines the network address of the backend service to which the TargetEndpoint forwards request messages.	N/A	No
`LoadBalancer`	Defines one or more named TargetServer configurations. Named TargetServer configurations can be used for load balancing defining 2 or more endpoint configuration connections. You can also use TargetServers to decouple API proxy configurations from concrete backend service endpoints URLs. See Load balance API traffic across multiple backend servers.	N/A	No
`Properties`	A set of optional HTTP configuration settings can be defined as properties of a TargetEndpoint. See Endpoint properties reference.	N/A	No
`FaultRules`	Defines how the TargetEndpoint reacts to an error. A fault rule specifies two items: · A Condition that specifies the fault to be handled based on the pre-defined category, subcategory, or name of the fault · One or more policies that define the behavior of the fault rule for the corresponding Condition	N/A	No
`DefaultFaultRule`	Handles any errors (system, transport, messaging or policy) that are not explicitly handled by another FaultRule. See Fault handling.	N/A	No
ScriptTarget			
`ResourceURL`	Defines the resource type (node) and the name of the main Node.js script that implements TargetEndpoint functionality. `<ResourceURL>node://server.js</ResourceURL>` ScriptTarget is an alternative to HTTPTargetConnection. (A TargetEndpoint can define either an HTTPTargetConnection or a ScriptTarget, but not both.)	N/A	Yes
`EnvironmentVariable`	Optionally pass environment variables to the main Node.js script.	N/A	No
`Arguments`	Optionally pass arguments to the main Node.js script.	N/A	N

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.3.12 TLS/SSL TargetEndpoint Configuration

TargetEndpoints often need to manage HTTP connections with heterogenous backend infrastructure. For this reason, a number of advanced configuration settings are supported.

Note:

Because SAP API Management Edge originally supported SSL, you will see some instances in the SAP API Management Edge UI and in the SAP API Management Edge XML that use the term "SSL". For example, the menu entry in the SAP API Management Edge UI that you use to view certs is called SSL Certificates. The XML tag that you use to configure a virtual host to use TLS is named`<SSLInfo>`.

TLS/SSL TargetEndpoint Configuration Elements

Name	Description	Default	Required?
SSLInfo			
`Enabled`	Indicates whether SSL is enabled for the Endpoint	true	Yes
`TrustStore`	A keystore containing trusted server certificates.	N/A	No
`ClientAuthEnabled`	A setting that turns on outbound client authentication (2-way SSL)	false	No
`KeyStore`	A keystore containing private keys used for outbound client authentication	N/A	Yes (if ClientAuthEnabled is true)
`KeyAlias`	The key alias of the private key used for outbound client authentication	N/A	Yes (if ClientAuthEnabled is true)

`Ciphers`	Supported ciphers for outbound SSL
	T o restrict ciphers, add the following elements listing the supported ciphers:
	`<Ciphers>`
	`<Cipher>TLS_RSA_WITH_3DES_EDE_CBC_SHA</Cipher>`
	`<Cipher>TLS_RSA_WITH_DES_CBC_SHA</Cipher>`
	`</Ciphers>`
		N/A	No

`Protocols`	Supported protocols for outbound TLS/SSL. If no protocols are specified, then all protocols available for the JVM will be permitted.
	To restrict protocols, add the following elements listing the supported protocols:
	`<Protocols>`
	`<Protocol>TLSv1</Protocol>`
	`<Protocol>TLSv1.2</Protocol>`
	`<Protocol>SSLv2Hello</Protocol>`
		N/A	No

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Default	Required?
	</Protocols>		

Sample TargetEndpoint with outbound client authentication enabled

```
<TargetEndpoint name="default">
  <HttpTargetConnection>
   <URL>https ://myservice.com</URL>
    <SSLInfo>
      <Enabled>true</Enabled>
      <ClientAuthEnabled>true</ClientAuthEnabled>
      <KeyStore>myKeystore</KeyStore>
      <KeyAlias>myKey</KeyAlias>
      <TrustStore>myTruststore</TrustStore>
    </SSLInfo>
  </HttpTargetConnection>
</TargetEndpoint>
```

For detailed instructions, refer to Configure client SSL from API Platform to your backend service

7.3.13 Using flow variables to set TLS/SSL values dynamically

You can also dynamically set TLS/SSL details to support flexible runtime requirements. For example, if your proxy connects to two potentially different targets (a test target and a production target), you can have your API proxy programmatically detect which environment it's calling and dynamically set references to the appropriate keystore and truststore.

In the following example of how the <SSLInfo> tag would be set in a TargetEndpoint configuration, the values can be supplied at runtime, for example, by a Java Callout, a JavaScript policy, or an Assign Message policy. Use whichever message variables contain the values you want to set.

Variables are allowed in only the following elements.

```
<SSLInfo>
    <Enabled>{myvars.ssl.enabled}</Enabled>
    <ClientAuthEnabled>{myvars.ssl.client.auth.enabled}</ClientAuthEnabled>
    <KeyStore>{myvars.ssl.keystore}</KeyStore>
    <KeyAlias>{myvars.ssl.keyAlias}</KeyAlias>
    <TrustStore>{myvars.ssl.trustStore}</TrustStore>
</SSLInfo>
```

Note:

Variable replacement of TLS/SSL values can only be used for a TargetEndpoint. Any attempt to use variables in any other context, such as virtual host configuration or message logging, will cause validation failures.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.3.14 Using references to set TLS/SSL values dynamically

When configuring a TargetEndpoint that uses HTTPS, you have to consider the case when the TLS/SSL cert expires, or a change to the system configuration requires you to update the cert. In an SAP API Management Edge installation, when configuring TLS/SSL by using static values or by using flow variables, there is a chance that you will have to restart the Message Processors.

However, you can optionally configure the TargetEndpoint to use a reference to the keystore or truststore instead. The advantage to using a reference is that you can update the reference to point to a different keystore or truststore to update the TLS/SSL cert without having to restart Message Processors.

Note:

You can only use a reference to the keystore and truststore; you cannot use a reference to the alias. When you change the reference to a keystore, ensure that the alias name of the cert is the same as in the old keystore.

For example, shown below is a TargetEndpoint that uses a reference to the keystore:

```
<SSLInfo>
    <Enabled>true</Enabled>
    <ClientAuthEnabled>false</ClientAuthEnabled>
    <KeyStore>ref://keystoreref</KeyStore>
    <KeyAlias>myKeyAlias</KeyAlias>
</SSLInfo>
```

Use the following POST API call to create the reference named keystoreref:

```
curl -X POST  -H "Content-Type:application/xml"
https://api.enterprise.sap.com/v1/o/{org_name}/e/{env_name}/references \
-d '<ResourceReference name="keystoreref">
    <Refers>myTestKeystore</Refers>
    <ResourceType>KeyStore</ResourceType>
</ResourceReference>' -u email:password
```

The reference specifies the name of the keystore and its type.

Use the following GET API call to view the reference:

```
curl -X GET
https://api.enterprise.sap.com/v1/o/[org_name}/e/{env_name}/references/keystoreref -u
uname:password
```

To later change the reference to point to a different keystore, ensuring that the alias has the same name, use the following PUT call:

```
curl -X PUT -H "Content-Type:application/xml"
https://api.enterprise.sap.com/v1/o/{org_name}/e/{env_name}/references/keystoreref \
-d '<ResourceReference name="keystoreref">
    <Refers>myNewKeystore</Refers>
    <ResourceType>KeyStore</ResourceType>
</ResourceReference>' -u email:password
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.3.15 TargetEndpoint with target load balancing

TargetEndpoints support load balancing across multiple named TargetServers using three load balancing algorithms.

For detailed instructions, refer to Load balance API traffic across multiple backend servers

7.3.16 Policies

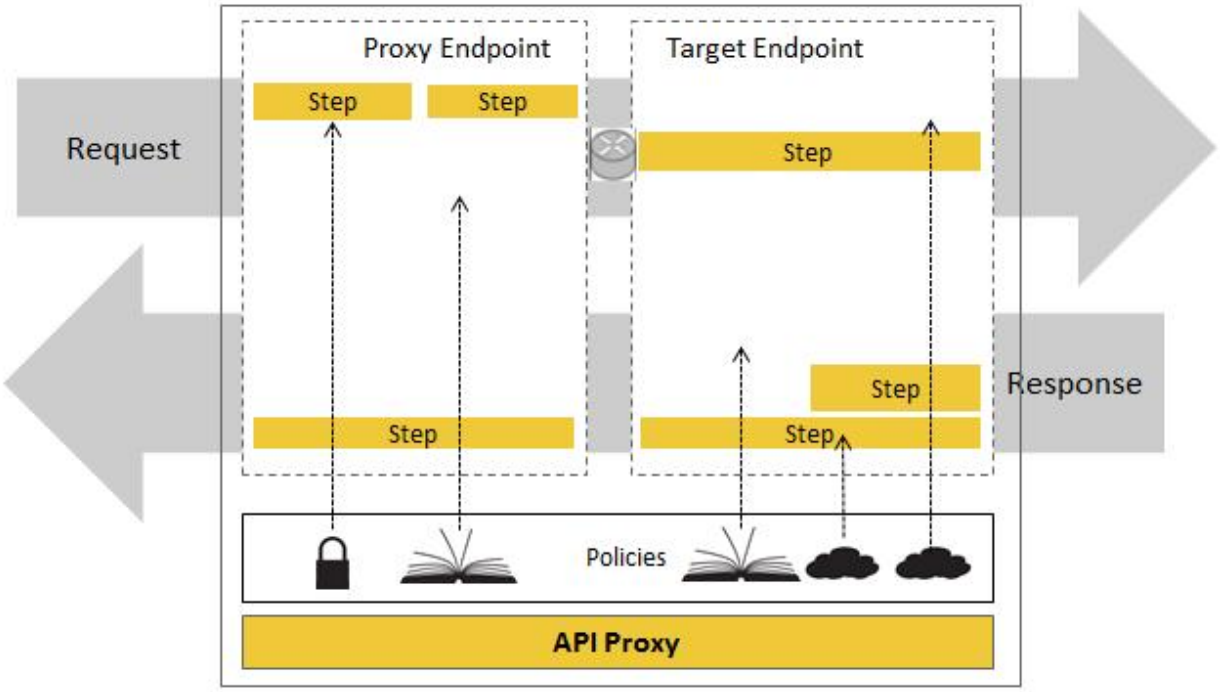
The /policies directory in an API proxy contains all policies available to be attached to Flows in the API proxy.

Policy Configuration Elements

Name	Description	Default	Required?
Policy			
`name`	The internal name of the policy. Characters you can use in the name are restricted to: `A-Z0-9._\-$ %`. However, the SAP API Management Edge UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the `<DisplayName>` element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Yes
`enabled`	Set to `true` to enforce the policy. Set to `false` to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	No
`continueOnError`	Set to `false` to return an error when a policy fails. This is expected behavior for most policies. Set to `true` to have flow execution continue even after a policy fails.	false	No
`async`	Note: This attribute does not make the policy execute asynchronously. In most cases, leave this with the default of `false`. When set to `true`, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to `true` improves API proxy performance. However, overusing async can hurt performance with too much thread switching.	false	No

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.3.17 Policy Attachment



As shown above, API proxy flows execute in the following sequence:

Policies are attached as processing steps to Flows. The policy's name is used to reference the policy to be enforced as a processing Step. The format of a policy attachment is the following:

```
<Step><Name>MyPolicy</Name></Step>
```

Policies are enforced in the order in which they are attached to a Flow. For example:

```
<Step><Name>FirstPolicy</Name></Step>
<Step><Name>SecondPolicy</Name></Step>
```

7.3.17.1 Policy Attachment Configuration Elements

Name	Description	Default	Required?
Step			
`Name`	The name of the policy to be executed by this Step definition.	N/A	Yes
`Condition`	A conditional statement that determines whether the policy is enforced or not. If a policy has an associated condition,	N/A	No

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Default	Required?
	then the policy only executes if the conditional statement evaluates to true.		

7.3.18 Flows

ProxyEndpoint and TargetEndpoint define a pipeline for request and response message processing. A processing pipeline consists of a request flow and a response flow. Each request flow and response flow is subdivided into a PreFlow, one or more optional 'conditional' or 'named' Flow, and a PostFlow.

- PreFlow: Always executes. Executes before any conditional Flows.
- PostFlow: Always executes. Executes after any conditional Flows.

Additionally, the ProxyEndpoint defines a PostClientFlow which executes after the request is returned to the requesting client app. Only MessageLogging policies can be attached to the PostClientFlow. The PostClientFlow is currently used primarily for measuring the time interval between the start and end timestamps for the response message.

Here's an example of a PostClientFlow with a message logging policy attached.

```
...
<PostFlow name="PostFlow">
    <Request/>
    <Response/>
</PostFlow>
<PostClientFlow>
    <Request/>
    <Response>
        <Step>
            <Name>Message-Logging-1</Name>
        </Step>
    </Response>
</PostClientFlow>
...
```

The API proxy processing pipeline executes Flows in the following sequence:

Request Pipeline:

1. Proxy Request PreFlow
2. Proxy Request Conditional Flows (Optional)
3. Proxy Request PostFlow
4. Target Request PreFlow
5. Target Request Conditional Flows (Optional)

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

6. Target Request PostFlow

Response Pipeline:

1. Target Response PreFlow
2. Target Response Conditional Flows (Optional)
3. Target Response PostFlow
4. Proxy Response PreFlow
5. Proxy Response Conditional Flows (Optional)
6. Proxy Response PostFlow
7. PostClientFlow Response (Optional)

Only those Flows with policy attachments need to be configured in ProxyEndpoint or TargetEndpoint configurations. PreFlow and PostFlow need only be specified in a ProxyEndpoint or TargetEndpoint configuration when a policy needs to be enforced during PreFlow or PostFlow processing.

In contrast to conditional flows, the ordering of PreFlow and PostFlow elements is not important--the API proxy will always execute each at the appropriate point in the pipeline, regardless of where they appear in the Endpoint configuration.

7.3.19 Conditional Flows

ProxyEndpoints and TargetEndpoints support an unlimited number of conditional flows (also known as 'named flows').

The API proxy tests for the condition specified in the conditional flow and, if the condition is met, the processing steps in the conditional flow are executed by the API proxy. If the condition is not met, then the processing steps in the conditional flow are bypassed. Conditional flows are evaluated in the order defined in the API proxy and the first one whose condition is met is executed.

By defining conditional flows, you gain the ability to apply processing steps in an API proxy based on:

- Request URI
- HTTP verb (GET/PUT/POST/DELETE)
- Value of a query param, header, and form param
- Many other types of conditions

For example, the following conditional statement defines an API resource `/accesstoken`. Any inbound request with the URI path suffix /accesstoken will cause this flow to be executed, along with any policies that are attached to the flow. Thus, this API resource functions as an OAuth token endpoint. If the inbound request URI does not include the path suffix /accesstoken, then the flow does not execute (although another conditional flow might).

```
<Flows>
  <Flow name="TokenEndpoint">
    <Condition>proxy.pathsuffix MatchesPath "/accesstoken"</Condition>
    <Request>
      <Step>
        <Name>GenerateAccessToken</Name>
      </Step>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    </Request>
  </Flow>
</Flows>
```

7.3.20 Flow Configuration Elements

Flow	A request or response processing pipeline defined by A ProxyEndpoint or TargetEndpoint		
`Name`	The unique name of the Flow.	N/A	Yes
`Condition`	A conditional statement that evaluates on or more variables to evaluate to true or false. All Flows other than the predefined PreFlow and PostFlow types must define a condition for their execution.	N/A	Yes
`Request`	The pipeline associated with Request message processing	N/A	No
`Response`	The pipeline associated with Response message processing	N/A	No

7.3.21 Step processing

The sequential ordering of conditional Flows is enforced by SAP API Management. Conditional Flows execute from top to bottom. The first conditional Flow whose condition evaluates to true is executed, and only one conditional Flow is executed.

For example, in the following Flow configuration, any inbound request that does not include the path `suffix /first or/second` will cause the ThirdFlow to execute, enforcing the policy called Return404.

```
  <Flows>
    <Flow name="FirstFlow">
      <Condition>proxy.pathsuffix MatchesPath "/first"</Condition>
      <Request>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
      <Step><Name>FirstPolicy</Name></Step>
    </Request>
  </Flow>
  <Flow name="SecondFlow">
    <Condition>proxy.pathsuffix MatchesPath "/second"</Condition>
    <Request>
      <Step><Name>FirstPolicy</Name></Step>
      <Step><Name>SecondPolicy</Name></Step>
    </Request>
  </Flow>
  <Flow name="ThirdFlow">
    <Request>
      <Step><Name>Return404</Name></Step>
    </Request>
  </Flow>
</Flows>
```

7.3.22 Resources

"Resources" are scripts, code, and XSL transformations that can be attached to Flows using policies.

Four resource types are supported:

- Compiled JavaScript: Stored under the `/jsc` directory
- Python scripts: Stored under the `/py` directory
- XSL transformations: Stored under the `/xsl` directory

Resources can be stored in an API proxy, an environment, or an organization. In each case, a resource is referenced by name in a Policy. API Platform resolves the name by moving from the API proxy, to environment to organization level.

A resource stored at the organization level can be referenced by Policies in any environment. A resource stored at the environment level can be referenced by Policies in that environment. A resource stored at the API proxy level can be referenced only by Policies in that API proxy.

7.4 Conditions reference

Conditions enable API proxies to behave dynamically at runtime. Conditions define operations on variables, which are evaluated by the SAP API Management processing pipeline. Conditional statements are boolean and always evaluate to true or false.

SAP API Management specifies a set of commonly used variables. For a complete list, see Variables reference. Custom variables can also be used in conditional statements. For instructions, see on setting custom variables; see Extract message content using ExtractVariables

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The basic structure of a conditional statement is:

```
<Condition>{variable.name}{operator}{"value"}</Condition>
```

For example:

```
<Condition>request.verb = "GET"</Condition>
```

Conditions can be chained. For example, the following condition evaluates to true only if the URI of the request matches /statuses/user_timeline.json and the HTTP verb of the request is GET.

```
<Condition>(proxy.pathsuffix MatchesPath "/statuses") and (request.verb =
"GET")</Condition>
```

Or:

```
(proxy.pathsuffix MatchesPath "/statuses") and (request.verb != "GET")
```

Conditional behavior is supported by three components of an API proxy:

Policy enforcement

Using conditional statements, you can control the enforcement of policies. A common use case is conditional transformation of response messages, based on HTTP header or message content.

For example, to conditionally transfrom XML to JSON based on the Accept header:

```
<Step>
  <Condition>request.header.accept = "application/json"</Condition>
  <Name>XMLToJSON</Name>
</Step>
```

Flow execution

Using conditional statements, you can control the execution of named flows in ProxyEndpoints and TargetEndpoints. Note that only 'named' flows can be executed conditionally. Preflows and postflows (both request and response) on ProxyEndpoints and TargetEndpoints execute for every transaction, and thus provide unconditional 'failsafe' capabilities.

For example, to execute a conditional request flow based on the HTTP verb of the request message, and conditional response flow based on a (potential) HTTP error code:

```
<Flow name="GetRequests">
  <Condition>response.verb = "GET"</Condition>
  <Request>
    <Step>
      <Condition>request.path MatchesPath "/statuses/**"</Condition>
      <Name>StatusesRequestPolicy</Name>
    </Step>
  </Request>
  <Response>
    <Step>
      <Condition>(response.status.code = 503) or (response.status.code =
400)</Condition>
      <Name>MaintenancePolicy</Name>
    </Step>
  </Response>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
</Flow>
```

Routes

Using conditional statements, you can control the target endpoint invoked by proxy endpoint configuration. A route rule forwards a request to a particular target endpoint. When more than one target endpoint is available, the route rule is evaluated for its condition and, if true, the request is forwarded to the named target endpoint.

For example, to conditionally route messages to designated target endpoints based on Content-Type:

```
<RouteRule name="target">
  <Condition>target-variable = "targets"</Condition>
  <TargetEndpoint>targets</TargetEndpoint>
</RouteRule>
<RouteRule name="default">
  <TargetEndpoint>default</TargetEndpoint>
</RouteRule>
```

See Create dynamic API flows using conditions for more information.

7.4.1 Path Expressions

Path expressions are used for matching URI paths., using "*" to represent a single path element and "**" to represent multiple URI levels.

For example:

Pattern	URIPath	values extracted
/*/a/{v1}	/x/y/z	no value extracted
/*/a/{v1}	/x/a/b	v1=b
/*/a/{v1}	/x/a/b/c/d	v1=b
/*/a/{v1}	/x/a/b;jsessionid=123456	v1=b
/*/a/{v1}	/x/y/z	no value extracted
/*/a/{v1}	/x/a/b	v1=b
/*/a/{v1}	/x/a/b/c/d	v1=b
/*/a/{v1}	/x/a/b;jsessionid=123456	v1=b
/a/**/feed/{v1}/{v2}	/a/b/feed/rss/1234	v1=rss v2=1234
/a/**/feed/{v1}/{v2}	/a/b/c/d/feed/rss/5678	v1=rss v2=5678
/a/**/feed/{v1}/{v2}	/a/b/c/feed/rss/5678/d/feed/atom/9876	v1=rss v2=9876

% is treated as escape character. The pattern `hello %{user%}{symbol} matches hello {user}!` but not hello world!.

Conditions can be categorized as follows:

- Operators: Wildcard patterns used in conditions.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- Relational operands: Evaluates whether a quantity is equal to, greater than, or less than another.
- Operands: The value of the conditions is used to determine the overall values.
- Literals: Literal values in a condition.

7.4.2 Operators

When using operators, observe the following restrictions:

- Operators cannot be used as variable names.
- A space character is required before and after an operator.
- To include an operator in a variable, a variable name must be enclosed in single quotes. For example,`'request.header.help!me'`.
- Arithmetic operators are supported `(+, *, -, /, %)`.
- SAP API Management relies on regular expressions as implemented in `java.util.regex`.

Symbol	In words (case insensitive)	Description
!	Not, not	Unary operator (takes a single input)
=	Equals, Is	Equals to
!=	NotEquals, IsNot	Not equals
:=	EqualsCaseInsensitive	Equals but is case insensitive
>	GreaterThan	Greater than
>=	GreaterThanOrEquals	Greater than or equal to
<	LesserThan	Lesser than
<=	LesserThanOrEquals	Lesser than or equal to
&&	And, and	And
||	Or	Or
(		Groups an expression
)		Closes an expression group
~~	JavaRegex	Matches a javax.util.regex compliant regular expression
~/	MatchesPath, LikePath	Matches a path expression
=|	StartsWith	Starts with

7.4.3 Relational operators and null operands

The following table shows the behavior when operands evaluate to null.

Operator	LHS null	RHS null	LHS and RHS null
=, ==, :=	false	false	true
=|	false	false	False
!=	true	true	False
>	true	false	False
>=	false	true	True
<	true	false	False
<=	true	false	True
~	false	N/A	False
!~	true	False	False
~/	false	N/A	false

7.4.4 Operands

Data types are defined for variables, so values can be adapted as needed for comparison. For example, the response status code = "404" is not a valid comparison because the response status code is an integer and not a string.

The system is configured to adapt operands to a common data type before comparing them. For example, if the response status code is 404, the expression response.status.code = "400" and the response.status.code = 400 are equivalent.

For numeric operands, the data type is interpreted as integer unless the value is terminated as follows:

- "f" or "F" (float, for example, 3.142f, 91.1F)
- "d" or "D" (double, for example, 3.142d, 100.123D)
- "l" or "L" (long, for example, 12321421312L)

In these cases, the system performs the adaptations shown in the following table.

> **Note**
>
> The dash character (-) in this table implies that the comparison is not performed, so the comparison is false.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

RHS LHS	Boolean	Integer	Long	Float	Double	String	Comparable	Object
Boolean	Boolean	Integer	Long	Float	Double	String		-
Integer	Integer	Integer	Long	Float	Double	String	Comparable	-
Long	Long	Long	Long	Float	Double	String	Comparable	-
Float	Float	Float	Float	Float	Double	String	Comparable	-
Double	Double	Double	Double	Double	Double	String	Comparable	-
String	String	String	String	String	String	String	Comparable	-
Comparable	Comparable	Comparable	Comparable	Comparable	Comparable	Comparable	Comparable	-
Object								

7.4.5 Literals

Null, true, and false are the literals available in conditions. For example:

- request.header.host is null
- flow.cachehit is true

Samples

```
<RouteRule name="default">
     <Condition>request.header.content-type = "text/xml"</Condition>
     <TargetEndpoint>XmlTargetEndpoint</TargetEndpoint>
</RouteRule>
-------------------------------------------------------------------------------
<Step>
    <Condition>response.status.code is 503</Condition>
    <Name>MaintenancePolicy</Name>
</Step>
-------------------------------------------------------------------------------
<Flow name="GetRequests">
    <Condition>response.verb="GET"</Condition>
    <Request>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
            <Step>
                <Condition>request.path ~ "/statuses/**"</Condition>
                <Name>StatusesRequestPolicy</Name>
            </Step>
        </Request>
        <Response>
            <Step>
                <Condition>(response.status.code = 503) or (response.status.code =
    400)</Condition>
                <Name>MaintenancePolicy</Name>
            </Step>
        </Response>
    </Flow>
```

7.5 Endpoint properties reference

7.5.1 TargetEndpoint Transport Properties

The HTTPTargetConnection element in TargetEndpoint configurations defines a set of HTTP transport properties. You can use these properties to set transport-level configurations.

Properties are set on TargetEndpoint HTTPTargetConnection elements as shown below:

```
    <TargetEndpoint name="default">
      <HTTPTargetConnection>
        <URL>http://weather.yahooapis.com</URL>
        <Properties>
          <Property name="supports.http10">true</Property>
          <Property name="request.retain.headers">User-Agent,Referer,Accept-
    Language</Property>
          <Property name="retain.queryparams">apikey</Property>
        </Properties>
      </HTTPTargetConnection>
    </TargetEndpoint>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.5.2 TargetEndpoint Transport Property Specification

Property Name	Default Value	Description
`keepalive.timeout.millis`	`60000`	Timeout for the target connection in the connection pool. if the connection is in the pool beyond the specified limit, then the connection is closed.
`connect.timeout.millis`	`3000`	Target connection timeout.
`io.timeout.millis`	`55000`	If there is no data to read for the specified number of milliseconds, or if the socket is not ready to write data for specified number of milliseconds, then the transaction is treated as a timeout. o If a timeout happens while reading HTTP request, `408,Request Timeout` is returned. o If a timeout happens while reading HTTP response, `504,Gateway Timeout` is returned. **Note** The io.timeout.millis value larger than 55000ms will be overridden by the default value of 55000ms set on the system. The increase of the timeout value on the system(more than 55000ms) cannot be performed because of the below reasons: o To wait for more than a minute for an API response would usually lead to bad experience for the end user. o It has a performance impact on the platform and this would compromise SLA guarantees.
`supports.http10`	true	If this is true and the client sends a 1.0 request, the target is also sent a 1.0 request. Otherwise 1.1 request is sent to target.
`supports.http11`	true	If this is true and the client sends a 1.1 request, the target is also sent a 1.1 request, otherwise 1.0 request is sent to target.
`use.proxy`	true	If set to true, and proxy configurations are specified in `http.properties` (on-premises deployments only), then target connections are set to use the specified proxy.
`use.proxy.tunneling`	true	If this is set to true, and proxy configurations are specified in `http.properties` (on-premises deployments only), then target connections are set to use the specified via tunnel. If the target uses SSL, then this property is ignored, and the message is always sent via a tunnel.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Property Name	Default Value	Description
enable.method.override	false	For the specified HTTP method, sets an X-HTTP-Method-Override header on the outbound request to the target service. For example, <Propertyname="GET.override.method">POST</Property>
*.override.method	N/A	For the specified HTTP method, sets an X-HTTP-Method-Override header on the outbound request. For example,<Propertyname="GET.override.method">POST</Property>
request.streaming.enabled	false	When true, HTTP request payloads are not read into buffer. The request payload streams as-is to the target endpoint. The request payload will not be available for policy enforcement in the Target Endpoint request flows.
response.streaming.enabled	false	When true, HTTP response payloads are not read into buffer. The response payload streams as-is to the client. The response payload will not be available for policy enforcement.
success.codes	N/A	By default, SAP API Management treats HTTP code 4XX or 5XX as errors, and it treats HTTP code 1XX, 2XX, 3XX as success. This property enables explicit definition of success codes, for example, 2XX,1XX, 505 treats any 100, 200 and 505 HTTP response codes as success.
compression.algorithm	N/A	By default, SAP API Management forwards requests to the target using the same compression type as the client request. If the request is received from client using, for example, gzip compression, then SAP API Management forwards the request to target using gzip compression. If the response received from target uses deflate, then SAP API Management forwards the response to the client using deflate. Supported values are: · gzip: always send message using gzip compression · deflate: always send message using deflate compression · none: always send message without any compression
request.retain.headers.enabled	true	By default, SAP API Management always retains all HTTP headers on outbound messages. When set to true, all HTTP headers present on the inbound request are set on the outbound request.
request.retain.headers	N/A	Defines specific HTTP headers from the request that should be set on the outbound request to the target service. For example, to 'passthrough' the User-Agent header, set the value ofrequest.retain.headers to User-Agent. Multiple HTTP

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Property Name	Default Value	Description
		headers are specified as a comma-separated list, for example, `User-Agent,Referer,Accept-Language`. This property overrides `request.retain.headers.enabled`. If `request.retain.headers`.enabled is set to false, any headers specified in the `request.retain.headersproperty` are still set on the outbound message.
`response.retain.headers.enabled`	true	By default, SAP API Management always retains all HTTP headers on outbound messages. When set to **`true`**, all HTTP headers present on the inbound response from the target service are set on the outbound response before it is passed to the ProxyEndpoint.
`response.retain.headers`	N/A	Defines specific HTTP headers from the response that should be set on the outbound response before it is passed to the ProxyEndpoint. For example, to 'passthrough' the `Expires` header, set the value of response.retain.headers to Expires. Multiple HTTP headers are specified as a comma-separated list, for example, `Expires,Set-Cookie`. This property overrides `response.retain.headers.enabled`. If `response.retain.headers.enabled` is set to `false`, any headers specified in the `response.retain.headers` property are still set on the outbound message.
`retain.queryparams.enabled`	true	By default, SAP API Management always retains all query parameters on outbound requests. When set to true, all query parameters present on the inbound request are set on the outbound request to the target service.
`retain.queryparams`	N/A	Defines specific query parameters to set on the outbound request. For example, to include the query parameter `apikey` from the request message, set `retain.queryparams to apikey`. Multiple query parameters are specified as a comma-separated list, for example, `apikey,environment`. This property overrides `retain.queryparams.enabled`.

7.5.3 ProxyEndpoint Transport Properties

ProxyEndpoint HTTPTargetConnection elements define a set of HTTP transport properties. These properties can be used to set transport-level configurations.

Properties are set on ProxyEndpoint HTTPProxyConnection elements as follows:

```
<ProxyEndpoint name="default">
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  <HTTPProxyConnection>
    <URL>http://weather.yahooapis.com</URL>
    <Properties>
      <Property name="allow.http10">true</Property>
      <Property name="request.streaming.enabled">true</Property>
    </Properties>
  </HTTPProxyConnection>
</ProxyEndpoint>
```

7.5.4 ProxyEndpoint Transport Property Specification

Property Name	Default Value	Description
`allow.http10`	true	If this is true and the client app sends a 1.0 request, the target is also sent a 1.0 request. If false, then an HTTP 1.1 request is sent to target.
`allow.http11`	true	If this is true and the client app sends a 1.1 request, the target is also sent a 1.1 request. If false, then an HTTP 1.0 request is sent to target.
`allow.http.method.*`	OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, PATCH	BY default, all HTTP methods are allowed. Use this setting To disable any HTTP method by setting the value for the property name, for example, name="allow.http.method.POST" to false. For example, `<Property name="allow.http.method.PUT">false</Property>` If the incoming request's method is not allowed, then the HTTP status code 405, "Method Not Allowed" will be returned, along with an Allow header whose value is the list of all methods that are allowed.
`X-Forwarded-For`	false	When set to true, the virtual host's PI address is added to the outbound request as the value of the `HTTP X-Forwarded-For` header.
`allow.POST.without.content.length`	false	When set to false, if the request method is POST and is not chunked, not compressed and the content-length header is missing, then the HTTP response code 411 "Length Required" is returned. When set to true, for the same scenario, the content-length is assumed to be zero, and the request is processed without parsing the payload.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Property Name	Default Value	Description
`allow.PUT.without.content.length`	false	When set to false case: If the request method is PUT and is not chunked, not compressed and the content-length header is missing, then the HTTP response code 411 "Length Required" is returned. When set to true, for the same scenario, the content-length is assumed to be zero, and the request is processed without parsing the payload.
`request.streaming.enabled`	false	By default, HTTP streaming is disabled, and the HTTP request payload is written to a buffer in memory before processing by the API proxy request pipeline. When set to true, the request payload is streamed without modification to the target, bypassing the API proxy request pipeline.
`response.streaming.enabled`	false	By default, HTTP streaming is disabled, and the HTTP response payload is written to a buffer in memory before processing by the API proxy response pipeline. When set to true, the response payload is streamed without modification to the client app, bypassing the API proxy response pipeline.
`compression.algorithm`	N/A	By default, SAP API Management honors the compression type set for any message received. For example, where a client submits a request that uses gzip compression, SAP API Management forwards the request to target using gzip compression. You can configure compression algorithms to be explicitly applied by setting this property on the TargetEndpoint or ProxyEndpoint. Supported values are gzip, deflate, and none. Use none to disable the default algorithm matching behavior of SAP API Management.

7.6 JavaScript Object Model

This topic discusses the SAP API Management Edge JavaScript Object Model. It's important you understand this model if you intend to use the JavaScript policy to add custom JavaScript to an API proxy.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.6.1 Background

As an API developer you are undoubtedly familiar with JavaScript. SAP API Management enables you to leverage your background with JavaScript to implement custom API behavior without having to change the code of your APIs or backend services.

SAP API Management can be used as a managed container for executing your custom JavaScript. You write the custom logic, and SAP API Management provides the container, security, management, monitoring, and analytics.

The benefit of this approach is that it lets you offload common functions like rate limiting, API key provisioning, OAuth authorization, transformations, which frees you to focus on coding the API behavior that is the most innovative and interesting.

You can combine SAP API Management's out-of-the-box policies with your own custom JavaScript to implement custom behavior securely and reliably.

This section is intended to be an overview and a reference.

7.6.2 Overview

If you have a good grasp of JavaScript, you just need to become familiar with SAP API Management's object model. JavaScript that executes in a browser relies on the "browser object model", or BOM. JavaScript that executes on SAP API Management relies on a message-oriented object model. This object model defines request, response, and context objects with associated properties. This section describes these objects and their properties.

SAP API Management relies on Rhino, which is an open-source implementation of JavaScript written in Java. SAP API Management also uses E4X, which is an extension of JavaScript that adds support for XML.

When your JavaScript is executed in an API proxy Flow, a scope is created for the execution.

A set of object references is created in the scope:

Name	Description	Properties
context	A wrapper for the message processing pipeline context and the request and response Flows that are executed by the ProxyEndpoint and TargetEndpoint.	flow, session
context.proxyRequest	An object that represents the inbound request message to the ProxyEndpoint (from the requesting app to the API proxy)	headers, query parameters, method, body, url
context.targetRequest	An object that represents the outbound request message from the TargetEndpoint (from the API proxy to the backend service).	headers, query parameters, method, body, url
context.targetResponse	An object that represents the inbound target response message (from the backend service to the API proxy)	headers, content, status

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Properties
context.proxyResponse	An object that represents the outbound proxy response message (from the API proxy to the requesting app)	headers, content, status

7.6.3 Context

The `context` object has global scope. It is available everywhere within the API proxy flow. It has four child objects: `proxyRequest, proxyResponse, targetRequest, targetResponse`. These child objects are scoped to the ambient request and response, either the proxy request and response or the target request and response. For example, if the JavaScript policy executes in the proxy endpoint part of the flow, then the context.proxyRequest and context.proxyResponse objects are in scope. If the JavaScript runs in a target flow, then the context.targetRequest and context.targetResponse objects are in scope.

The `context` object also has properties and methods, which are described in detail in this topic. For example, the following JavaScript code example uses the context.flow property and calls the get/setVariable() methods on context.

```
if (context.flow=="PROXY_REQ_FLOW") {
     var username = context.getVariable("request.formparam.user");
     context.setVariable("USER.name", username);
}
```

These methods interact directly with flow variables. The context.flow property value is the current flow scope. In the proxy request flow, it's set to the constant `PROXY_REQ_FLOW`. If in the target response flow, it's set to `TARGET_RESP_FLOW`. This constant is handy for executing scope-specific code. The getter lets you get flow variables and the setter lets you set flow variables. These variables are generally available in the proxy flow and can be consumed by other policies.

7.6.4 The request and response objects

The `request` and `response` objects are "shorthand" references to the ambient request and response, either the proxy request and response or the target request and response. The objects these variables refer to depend upon the context in which the JavaScript policy executes. If the JavaScript runs in the flow of a proxy endpoint, then the request and response variables refer to context.proxyRequest and context.ProxyResponse. If the JavaScript runs in a target flow, then the variables refer to the context.targetRequest and context.targetResponse.

7.6.5 The print() function

The JavaScript object model includes a print() function that you can use to output debug information to the SAP API Management Edge Trace tool.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.6.6 crypto object reference

The crypto object lets you perform basic cryptographic hashing functions in JavaScript.

Note:

The crypto functions support hashing only -- there is no support for 2-way encryption.

The crypto object has global scope. It is available everywhere within the API proxy flow. Crypto lets you work with these hash objects:

- SHA-1
- SHA256
- SHA512
- MD5

Note:

SAP API Management recommends that you use the crypto object to perform basic cryptographic functions in your JavaScript programs running on SAP API Management Edge.

7.6.6.1 Working with SHA-1 objects

You can create SHA-1 objects, update them, and convert them to hex and base64 values.

Create a new SHA-1 object

```
var _sha1 = crypto.getSHA1();
```

Update an SHA-1 object

Syntax

```
_sha1.update(value);
```

Parameters

- value - (String) Any string value.

Example

Update an SHA-1 object:

```
_sha1.update("salt_value");
_sha1.update("some text");
```

Return the SHA-1 object as a hex string

```
var _hashed_token = _sha1.digest();
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Return the SHA-1 object as a base64 string

```
var _hashed_token = _sha1.digest64();
```

7.6.6.2 Working with SHA-256 objects

You can create SHA-256 objects, update them, and convert them to hex and base64 values.

Create a new SHA-256 object

```
var _sha256 = crypto.getSHA256();
```

Update an SHA-256 object

Syntax

```
_sha256.update(value);
```

Parameters

- value - (String) Any string value.

Example

Update an SHA-256 object:

```
_sha256.update("salt_value");

_sha256.update("some text");
```

Return the SHA-256 object as a hex string

```
var _hashed_token = _sha256.digest();
```

Return the SHA-256 object as a base64 string

```
var _hashed_token = _sha256.digest64();
```

7.6.6.3 Working with SHA-512 objects

You can create SHA-512 objects, update them, and convert them to hex and base64 values.

Create a new SHA-512 object

```
var _sha512 = crypto.getSHA512();
```

Update an SHA-512 object

Syntax

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
_sha512.update(value);
```

Parameters

- value - (String) Any string value.

Example

Update an SHA-512 object:

```
_sha512.update("salt_value");

_sha512.update("some text");
```

Return the SHA-512 object as a hex string

```
var _hashed_token = _sha512.digest();
```

Return the SHA-512 object as a base64 string

```
var _hashed_token = _sha512.digest64();
```

7.6.6.4 Working with MD5 objects

You can create MD5 objects, update them, and convert them to hex and base64 values.

Create a new MD5 object

```
var _md5 = crypto.getMD5();
```

Update an MD5 object

Syntax

```
_md5.update(value);
```

Parameters

- value - (String) Any string value.

Example

Update an MD5 object:

```
_md5.update("salt_value");

_md5.update("some text");
```

Return the MD5 object as a hex string

```
var _hashed_token = _md5.digest();
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Return the MD5 object as a base64 string

```
var _hashed_token = _md5.digest64();
```

7.6.6.5 Crypto date / time support

The crypto object supports date / time functions.

`crypto.dateFormat()`

Returns a date in string format.

Syntax

```
crypto.dateFormat(format, [timezone], [time])
```

Parameters

- format - (String) A string-formatted date. For example: 'YYYY-MM-DD HH:mm:ss.SSS'
- timezone - (String, optional) A time zone. Default: UTC
- time - (Number, optional) A Unix timestamp value to format. Default: current time

Examples

Get the current time, down to milliseconds:

```
var _now = crypto.dateFormat('YYYY-MM-DD HH:mm:ss.SSS');
```

Get the current time for the Pacific Time Zone:

```
var _pst = crypto.dateFormat('YYYY-MM-DD HH:mm:ss.SSS','PST');
```

Get the value of ten seconds from now:

```
var _timeNow = Number(context.getVariable('system.timestamp'));
var ten_seconds = crypto.dateFormat('YYYY-MM-DD HH:mm:ss.SSS','PST', _timeNow + 10 *
1000);
```

7.6.6.6 Use getHash() to get any of the supported hash objects

Examples

```
var _hash1 = crypto.getHash('MD5');

var _hash2 = crypto.getHash('SHA-1');

var _hash3 = crypto.getHash('SHA-256');
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
var _hash4 = crypto.getHash('SHA-512');
```

7.6.6.7 Sample with crypto

```
try {
    //get values to use with hash functions
    var salt = context.getVariable("salt") || 'SomeHardCodedSalt';
    var host = context.getVariable("request.header.Host");
    var unhashed_token = "";

    var _timeNow = Number(context.getVariable('system.timestamp'));
    var now = crypto.dateFormat('YYYY-MM-DD HH:mm:ss.SSS','PST', _timeNow);
    unhashed_token = "|" + now + "|" + host

    //generate a hash with the unhashedToken:
    var sha512 = crypto.getSHA512();
    sha512.update(salt);
    sha512.update(unhashed_token);

    //convert to base64
    var base64_token = sha512.digest64();

    // set headers
    context.setVariable("request.header.now", now);
    context.setVariable("request.header.token", base64_token);

} catch(e) {
    throw 'Error in Javascript';
}
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.6.7 Context methods

`getVariable():` Retrieves the value of a pre-defined or custom variable.

For example, to get the value for the current year:

```
var year = context.getVariable('system.time.year');
```

`setVariable():` Sets the value for a custom variable or for any writable pre-defined variables.

```
var org = context.setVariable('organization.name.myorg', value);
```

A common scenario for setting a variable is when an API proxy must dynamically write the target URL. The following JavaScript obtains the value of a variable called WOEID.location, appends that value as a query parameter to the:

URL `http://weather.yahooapis.com/forecastrss?w=`, and then sets the target.url.

`WOEID.location` is a custom variable. `target.url.target.url` is a pre-defined variable.

```
context.setVariable("target.url",
"http://weather.yahooapis.com/forecastrss?w="+context.getVariable("WOEID.location")
);
```

> **Note**
>
> Any writable pre-defined variable and any custom variable can be dynamically set from JavaScript. For a complete list of pre-defined variables, see Variables reference.

`removeVariable():` Removes a variable from the context.

```
var org = context.removeVariable('organization.name.myorg');
```

7.6.8 Context Properties

`flow`: The name of the current Flow

The flow property is a string that identifies the current Flow. This property is used to indicate the Flow to which the JavaScript is attached. Supported values are:

- `PROXY_REQ_FLOW`
- `PROXY_RESP_FLOW`
- `TARGET_REQ_FLOW`
- `TARGET_RESP_FLOW`

Each Flow name encompasses the PreFlow, PostFlow, and any conditional Flows defined in the ProxyEndpoint(s) or TargetEndpoint(s).

This optional property is useful when common JavaScript is executed in more than one Flow , but might vary its behavior depending on the Flow in which it executes. Use the Flow property for JavaScript modules intended to be reused in multiple API proxies, in which the code is required to check the current Flow before executing logic.

For example:

Set an HTTP header only on the targetRequest Flow:

```
if (context.flow=="TARGET_REQ_FLOW") {
     context.targetRequest.headers['TARGET-HEADER-X]='foo';
}
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Set the content only on the proxyResponse Flow:

```
if (context.flow=="PROXY_RESP_FLOW") {
     context.proxyResponse.content='bar';
}
```

session: A map of name/value pairs that can be used to pass objects between two policies executing within the same message context

For example, you can set the context as follows:

```
context.session['key']  = 123;
```

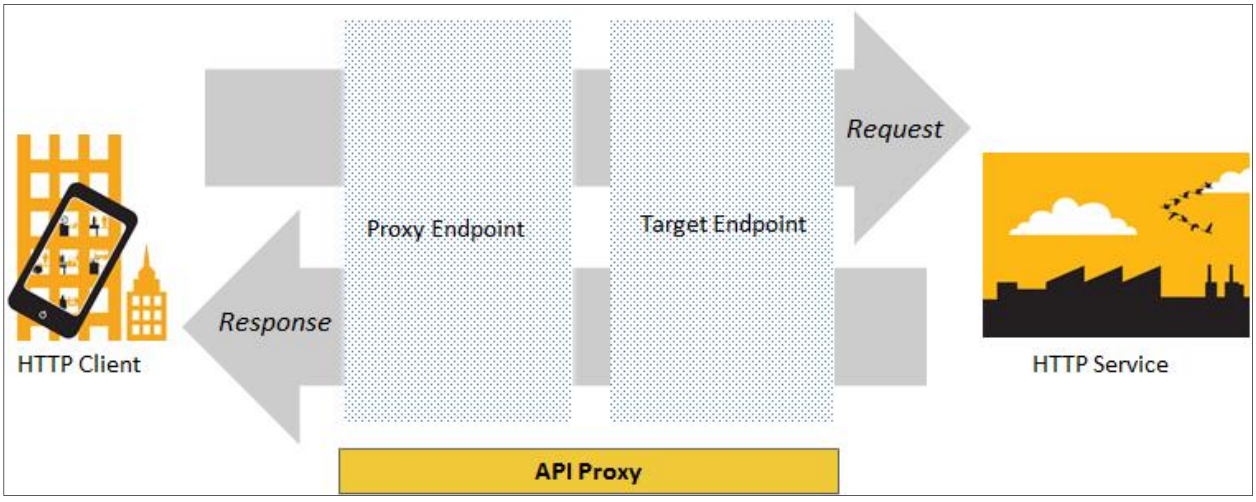
When needed you can get the context as follows::

```
var value = context.session['key']; // 123
```

7.6.9 Messages

As shown below, a complete API proxy Flow encompasses four distinct phases, each of which has an associated message object:

- `proxyRequest`: The inbound request message received from the requesting client.
- `targetRequest`: The outbound request message sent to the backend service.
- `proxyResponse`: The outbound response message returned to the requesting client.
- `targetResponse`: The inbound request message received from the backend service.



7.6.9.1 Request messages

The context provides access to objects that represent request messages. For each HTTP transaction the executes in an API proxy, two request message objects are created: one 'inbound' (the request from the client) and one 'outbound' (the request generated by the API proxy and submitted to the backend target.)

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

These two request objects are called proxyRequest and targetRequest, respectively. The proxyRequest and targetRequest objects enable your JavaScript to specify which message object within the context should be the target of a JavaScript operation.

Request object properties

Property name	Description
url	The `url` property is a read/write convenience property that combines scheme, host, port, path and query parameters for the targetRequest. The complete URL of the request is composed of the following properties: · protocol: The protocol of the URL (for example, HTTP, HTTPS) · port: The port (for example, :80, :443) · host: The host of the URL (for example, www.example.com) · path: The path of the URI (for example, /v1/weather) When getting `url`, a URL is returned in the following format: `protocol://host:port/path?queryParams`

	Examples
	`context.targetRequest.url = 'http://www.example.com/path?q1=1'`
	`context.targetRequest.protocol ='https';`

headers	HTTP request headers as a mapping of String => List

	Examples
	For this HTTP request:
	`POST /v1/blogs HTTP/1.1`
	`Host: api.example.com`
	`Content-Type: application/json`
	`Authorization: Bearer ylSkZIjbdWybfs4fUQe9BqP0LH5Z`
	The following JavaScript:
	`context.proxyRequest.headers['Content-Type'];`
	`context.proxyRequest.headers['Authorization'];`
	will return the following values
	`application/json`
	`Bearer ylSkZIjbdWybfs4fUQe9BqP0LH5Z`

queryParameters	The request message query parameters as a mapping of String => List.

	Examples
	`"?city=PaloAlto&city=NewYork"`
	can be accessed as:
	`context.proxyRequest.queryParams['city'];  // == 'PaloAlto'`
	`context.proxyRequest.queryParams['city'][0]  // == 'PaloAlto'`
	`context.proxyRequest.queryParams['city'][1];  // == 'NewYork'`
	`context.proxyRequest.queryParams['city'].length; // == 2`

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Property name	Description
method	The HTTP verb (GET, POST, PUT, DELETE. PATCH, etc.) associated with the request

	Examples
	For this request:
	`POST /v1/blogs HTTP/1.1`
	`Host: api.example.com`
	`Content-Type: application/json`
	`Authorization: Bearer ylSkZIjbdWybfs4fUQe9BqP0LH5Z`
	The following JavaScript:
	`context.proxyRequest.method;`
	will return the following value
	`POST`

body	The message body (payload) of the HTTP request.
	The request body has the following members:
	`context.targetRequest.body.asXML;`
	`context.targetRequest.body.asJSON;`
	`context.targetRequest.body.asForm;`

	Examples
	For an XML body:
	`<customer number='1'>`
	`<name>Fred<name/>`
	`<customer/>`
	To access the elements of the XML object as follows:
	`var name = context.targetRequest.asXML.name;`
	To access XML attributes attributes, use the @ notation.
	`var number = context.targetRequest.body.asXML.@number;`
	For a JSON request body:
	`{`
	`"a":  1 ,`
	`"b" : "2"`
	`}`
	`var a = context.proxyRequest.body.asJSON.a;    // == 1`
	`var b = context.proxyRequest.body.asJSON.b;    // == 2`
	To read form parameters:
	`"vehicle=Car&vehicle=Truck"`
	`-------------------------------------------------`
	`v0 = context.proxyRequest.body.asForm['vehicle'][0];`
	`v1 = context.proxyRequest.body.asForm['vehicle'][1];`

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.6.9.2 Response messages

SAP API Management generates two response objects: one for the ProxyEndpoint and one for the TargetEndpoint.

> **Note**
>
> Before the targetRequest PreFlow has executed, the response object has a value of undefined.

Response object properties

Property name	Description
headers	The HTTP headers of the response message as a mapping of String => List.
	Examples `var cookie = context.targetResponse.headers['Set-Cookie'];`
status	The status code with status message as a property. Both status code and status message are available as properties.

	Examples
	`var status = context.targetResponse.content.status; // 200`
	`var msg = context.targetResponse.content.status.message; // "OK"`

content	The HTTP body (payload content) of the response message.
	Response content has the following members:
	`context.targetResponse.content.asXML;`
	`context.targetResponse.content.asJSON;`

	Examples:
	For an XML response:
	`"<customer number='1'>`
	`<name>Fred<name/>`
	`<customer/>"`
	You can access the elements of the XML object as follows:
	`var name = context.proxyResponse.content.asXML.name;`
	You can access the attributes of the XML object using the @ sign notation.
	`var number = context.proxyResponse.content.asXML.@number;`
	For JSON response:
	`{`
	`"a": "1" ,`
	`"b": "2"`
	`}`
	`var a = context.targetResponse.content.asJSON.a; // == 1`
	`var b = context.targetResponse.content.asJSON.b; // == 2`

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.6.9.3 HttpClient

The JavaScript HTTP Client can be used to make multiple parallel, asynchronous HTTP requests to any URL. This makes it useful for developing composite services, as well as for consolidating multiple backend calls into a single API method. You can use the HTTP client as an alternative to the ServiceCallout policy.

The HTTP Client exposes two methods: get() and send().

`get()`

A convenience method for simple HTTP GETs, with no support for HTTP headers.

Example:

```
var exchange = httpClient.get("http://www.example.com");
```

`send()`

Enables full configuration of the request message using the request object to contain the properties of the HTTP request.

The httpClient request object is identical in type to the request object generated by a standard Flow.

```
var myRequest = new Request();
myRequest.url = "http://www.example.com";
var exchange = httpClient.send(myRequest);
```

Alternatively:

```
var headers = {'X-SOME-HEADER' : 'some value' };
var myRequest = new Request("http://www.example.com","GET",headers);
var exchange = httpClient.send(myRequest);
```

or

```
var headers = {'Content-Type : 'application/xml' };
   var myRequest = new Request("http://www.example.com","POST",headers,"");
       var exchange = httpClient.send(myRequest);
```

The calls to get() and send() immediately return an object that can used later to get the actual HTTP response, or to check whether the response has timed out.

Example:

```
var ex1 = httpClient.get("http://www.example.com?api1");
var ex2 = httpClient.get("http://www.example.com?api2");
```

The returned object can be accessed later during flow processing, for example by another JavaScript policy:

```
ex1.waitForComplete();      // Thread is paused until the response is returned or
error or step time limit has been reached.
ex2.waitForComplete(100);   // Thread is paused for a maximum of 100 ms.

if (ex1.isSuccess() && ex2.isSuccess() {
 response.content = ex1.getResponse().content +
            ex2.getResponse().content
}
```

A call to httpClient.send() returns an exchange object. The exchange object object has no properties, and it exposes the following methods:

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- `isError()`: Returns true if the httpClient was unable to connect to the server. HTTP status codes 4xx and5xx result in isError() false, as the connection completed and a valid response code was returned. IfisError() returns true, then a call to getResponse() returns the JavaScript undefined.
- `isSuccess()`: returns true if the send was complete and successful.
- `isComplete()`: returns true if the request is complete.
- `waitForComplete()`: pauses the thread until the requests is complete (by success or error).
- `getResponse()`: returns the response object if the httpClient.send() was complete and successful. The httpClient response object is identical in type to the request object generated by a standard Flow.
- `getError()`: If the call to httpClient.send() resulted in an error, returns the error message as a string.

For example, to track the status of an HTTP request:

```
function userCheck() {
  var url = getAppServicesUrl() + '/users/' + username,
      headers = {
        Authorization : 'Bearer ' + appServicesAccessToken
      },
      req = new Request(url, 'GET', headers),
      exchange = httpClient.send(req),
      response, status;

  // Wait for the asynchronous GET request to finish
  exchange.waitForComplete();

  // get the response object from the exchange
  response = exchange.getResponse();

  // get the HTTP status code from the response
  status = response.status;

  if (status == 200) {
    context.setVariable('userCheck.trace', 'user exists');
  }
  else if (status == 404) {
    context.setVariable('userCheck.trace', 'user does not exist');
```

For example, to use httpClient to make a call to retrieve an OAuth access token:

```
function getAccessToken() {
  var bodyObj = {
    'grant_type': translatorApi.grantType,
    'scope': translatorApi.scopeUrl,
    'client_id': translatorApi.clientId,
    'client_secret': translatorApi.clientSecret
  };
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  var req = new Request(translatorApi.authUrl, 'POST', {},
serializeQuery(bodyObj));
  var exchange = httpClient.send(req);

  // Wait for the asynchronous POST request to finish
  exchange.waitForComplete();

  if (exchange.isSuccess()) {
    var responseObj = exchange.getResponse().content.asJSON;

    if (responseObj.error) {
      throw new Error(responseObj.error_description);
    }

    return responseObj.access_token;
  } else if (exchange.isError()) {
    throw new Error(exchange.getError());
  }
}
```

7.6.9.4 Policy attachment

JavaScript is attached to API proxy Flow configuration using the Javascript Policy type.

See Customize an API using JavaScript.

7.7 OAuth Error Code Reference

7.7.1 Authorization Code

Invalid Redirect URI

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Invalid redirection uri
http://www.invalid_example.com"}
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

No Redirect URI

```
HTTP/1.1 400 Bad Request {"ErrorCode" : "invalid_request", "Error" :"Redirection
URI is required"}
```

Invalid Key

```
HTTP/1.1 401 Unauthorized {"ErrorCode" : "invalid_request", "Error" :"Invalid
client id : AVD7ztXReEYyjpLFkkPiZpLEjeF2aYAz. ClientId is Invalid"}
```

Missing Key

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"The request is missing a required
parameter : client_id"}
```

Invalid Response Type

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Response type must be code"}
```

Missing Response Type

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"The request is missing a required
parameter : response_type"}
```

7.7.2 Generate AccessToken

Invalid Auth Code

```
HTTP/1.1 404 Not Found
{"fault":{"faultstring":"Invalid Authorization
Code","detail":{"errorcode":"keymanagement.service.invalid_request-
authorization_code_invalid"}}}
```

No Redirect URI

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Required param : redirect_uri"}
```

Invalid Redirect URI

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Invalid redirect_uri : oob"}
```

Invalid Client ID

```
HTTP/1.1 401 Unauthorized
{"ErrorCode" : "invalid_client", "Error" :"Client identifier is required"}
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

No Client ID

```
HTTP/1.1 401 Unauthorized
{"ErrorCode" : "invalid_client", "Error" :"Client identifier is required"}
```

Invalid GrantType

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Unsupported grant type :
client_credentials_invalid"}
```

No Username

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Required param : username"}
```

No Password

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Required param : password"}
```

No GrantType (Custom Policy)

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Required param : grant_type"}
```

No AuthCode

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Required param : code"}
```

7.7.3 Implicit

Invalid Client ID

```
HTTP/1.1 401 Unauthorized
{"ErrorCode" : "invalid_request", "Error" :"Invalid client id :
AVD7ztXReEYyjpLFkkPiZpLEjeF2aYAz. ClientId is Invalid"}
```

No Client ID

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"The request is missing a required
parameter : client_id"}
```

Invalid Response Type

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Response type must be token"}
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

No Response Type

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"The request is missing a required
parameter : response_type"}
```

Invalid Redirect URI

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Invalid redirection uri
http://www.invalid_example.com"}
```

No Redirect URI

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Redirection URI is required"}
```

7.7.4 Refresh Token

Invalid RefreshToken

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Invalid Refresh Token"}
```

Invalid Scope

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Invalid Scope"}
```

Invalid Client ID

```
HTTP/1.1 401 Unauthorized
{"ErrorCode" : "invalid_client", "Error" :"Client identifier is required"}
```

No Client ID

```
HTTP/1.1 401 Unauthorized
{"ErrorCode" : "invalid_client", "Error" :"Client identifier is required"}
```

7.7.5 Verify AccessToken

Invalid AccessToken

```
HTTP/1.1 401 Unauthorized
{"fault":{"faultstring":"Invalid Access
Token","detail":{"errorcode":"keymanagement.service.invalid_access_token"}}}
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Invalid Resource

```
HTTP/1.1 401 Unauthorized
{"fault":{"faultstring":"APIResource \/facebook\/acer does not
exist","detail":{"errorcode":"keymanagement.service.apiresource_doesnot_exist"}}}
```

Invalid Scope

```
HTTP/1.1 403 Forbidden
{"fault":{"faultstring":"Required scope(s) :
VerifyAccessToken.scopeSet","detail":{"errorcode":"steps.oauth.v2.InsufficientScope
"}}}
```

No Auth Header

```
HTTP/1.1 401 Unauthorized
{"fault":{"faultstring":"Invalid access
token","detail":{"errorcode":"oauth.v2.InvalidAccessToken"}}}
```

No match for ApiProduct (With Env & Proxy Configured)

```
HTTP/1.1 401 Unauthorized
{"fault":{"faultstring":"Invalid API call as no apiproduct match
found","detail":{"errorcode":"keymanagement.service.InvalidAPICallAsNoApiProductMat
chFound"}}}
```

7.8 OAuth Flow Variables

SAP API Management encapsulates OAuth 1.0a and OAuth 2 capabilities in a set of policies. The life cycle management of tokens and secrets, including generation, validation, and storage, is managed by SAP API Management on behalf of your backend services.

This section specifies the flow variables defined by OAuth policies. The variables can be used to implement custom behavior for OAuth flows. For OAuth usage, see OAuth.

7.8.1 OAuth 2.0 Flow Variables

The flow variables defined in this table are populated when the respective OAuth policies are executed, and hence are available to other policies or applications executing in the API proxy flow.

Verify access token policy
· organization_name · organization_name · developer.id · developer.app.name · client_id · grant_type

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Verify access token policy

- token_type
- access_token
- accesstoken.{custom_attribute}
- issued_at
- expires_in
- status
- scope
- apiproduct.name*
- apiproduct.<custom_attribute_name>*

API product variables

Sample policy:

```
<OAuthV2 name="VerifyAccessToken">
    <Operation>VerifyAccessToken</Operation>
    <Scope>space-separated-scopes</Scope>*
    <AccessToken>flow.variable</AccessToken>*
    <AccessTokenPrefix>Bearer</AccessTokenPrefix>*
</OAuthV2>
```

Only bearer tokens are supported. MAC tokens are not supported.

By default, the access token must be passed in the Authorization HTTP request header.

For example:

`"Authorization: Bearer {PlainText_AccessToken}"`

Any `flow.variable` can be passed as:

- `HTTP Header: request.header.{variable_name}`
- `Query parameter: request.queryparam.{variable_name}`
- `Form parameter: request.formparam.{variable_name}`

If the optional fields are not specified, the values are extracted per the OAuth 2.0 specification.

Generate authorization code policy

Variables set on success:

- oauthv2authcode.{policy_name}.code
- oauthv2authcode.{policy_name}.redirect_uri
- oauthv2authcode.{policy_name}.scope
- oauthv2authcode.{policy_name}.client_id

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Sample policy:

```
<OAuthV2 name="GetAuthCode">
     <Operation>GenerateAuthorizationCode</Operation>
     <ExpiresIn>1000<ExpiresIn>
     <ResponseType>flow.variable</ResponseType>*
     <ClientId>flow.variable</ClientId>*
     <RedirectUri>flow.variable</RedirectUri>*
     <Scope>flow.variable</Scope>*
     <State>flow.variable</State>*
     <Attributes>*
       <Attribute name=”1” ref=”flow.variable”>value1</Attribute>
       <Attribute name=”2” ref=”flow.variable”>value2</Attribute>
     </Attributes>
</OAuthV2>
```

* Optional

Any `flow.variable` can be passed as:

- `HTTP Header: request.header.{variable_name}`
- `Query parameter: request.queryparam.{variable_name}`
- `Form parameter: request.formparam.{variable_name}`

If the optional fields are not specified, the values are extracted per the OAuth 2.0 specification.

Attribute values are derived dynamically from the specified flow variable, or statically using a default value in the policy.

If both are specified, flow variable takes precedence.

Generate access token policy for grant types authorization code, user credentials, and client credentials

Variables set on success:

- `oauthv2accesstoken.{policy_name}.access_token`
- `oauthv2accesstoken.{policy_name}.token_type`
- `oauthv2accesstoken.{policy_name}.expires_in`
- `oauthv2accesstoken.{policy_name}.refresh_token`

Sample policy:

```
<OAuthV2 name="GenerateAccessToken">
    <Operation>GenerateAccessToken</Operation>
     <ExpiresIn>1000<ExpiresIn>
     <SupportedGrantTypes>*
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Sample policy:

```
        <GrantType>authorization_code</GrantType>
        <GrantType>password</GrantType>
       <GrantType>client_credentials</GrantType>
      </SupportedGrantTypes>
      <GrantType>flow.variable</GrantType>*
      <ClientId>flow.variable</ClientId>*
      <RedirectUri>flow.variable</RedirectUri>*
      <Scope>flow.variable</Scope>*
      <AppEndUser>flow.variable</AppEndUser>*
      <Code>flow.variable</Code>*
      <UserName>flow.variable</UserName>*
      <PassWord>flow.variable</PassWord>*
      <Attributes>*
        <Attribute name="1" ref="flow.variable">value1</Attribute>
        <Attribute name="2" ref="flow.variable">value2</Attribute>
      </Attributes>
</OAuthV2>
```

* Optional

Any `flow.variable` can be passed as:

- `HTTP Header: request.header.{variable_name}`
- `Query parameter: request.queryparam.{variable_name}`
- `Form parameter: request.formparam.{variable_name}`

If the optional fields are not specified, the values are extracted per the OAuth 2.0 specification.

Attribute values are derived dynamically from the specified flow variable, or statically using a default value in the policy.

If both are specified, flow variable takes precedence.

Generate access token policy for Implicit grant type

Variables set on success:

- `oauthv2accesstoken.{policy_name}.access_token`
- `oauthv2accesstoken.{policy_name}.token_type`
- `oauthv2accesstoken.{policy_name}.expires_in`
- `oauthv2accesstoken.{policy_name}.refresh_token`

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Sample policy:

```
<OAuthV2 name="GenerateAccessToken">
     <Operation>GenerateAccessTokenImplicitGrant</Operation>
     <ExpiresIn>1000<ExpiresIn>
     <ResponseType>flow.variable></ResponseType>*
     <ClientId>flow.variable></ClientId>*
     <RedirectUri>flow.variable></RedirectUri>*
     <Scope>flow.variable></Scope>*
     <State>flow.variable></State>*
     <AppEndUser>flow.variable</AppEndUser>*
     <Attributes>*
       <Attribute name=”1” ref=”flow.variable”>value1</Attribute>
       <Attribute name=”2” ref=”flow.variable”>value2</Attribute>
     </Attributes>
</OAuthV2>
```

* Optional

Any `flow.variable` can be passed as:

- `HTTP Header: request.header.{variable_name}`
- `Query parameter: request.queryparam.{variable_name}`
- `Form parameter: request.formparam.{variable_name}`

If the optional fields are not specified, the values are extracted per the OAuth 2.0 specification.

Attribute values are derived dynamically from the specified flow variable, or statically using a default value in the policy.

If both are specified, flow variable takes precedence.

Refresh access token policy

Variables set on success:

- `oauthv2accesstoken.{policy_name}.access_token`
- `oauthv2accesstoken.{policy_name}.token_type`
- `oauthv2accesstoken.{policy_name}.expires_in`
- `oauthv2accesstoken.{policy_name}.refresh_token`

Sample policy:

```
<OAuthV2 name="RefreshAccessToken">
  <Operation>RefreshAccessToken</Operation>
  <ExpiresIn>1000<ExpiresIn>
  <GrantType>flow.variable</GrantType>*
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Sample policy:

```
  <RefreshToken>flow.variable</RefreshToken>*
</OAuthV2>
```

* Optional

Any `flow.variable` can be passed as:

- `HTTP Header: request.header.{variable_name}`
- `Query parameter: request.queryparam.{variable_name}`
- `Form parameter: request.formparam.{variable_name}`

If the optional fields are not specified, the values are extracted per the OAuth 2.0 specification.

Get client attributes policy

Sample policy:

```
<GetOAuthV2Info name="GetClientAttributes">
<ClientId ref="{variable_name}"/>
</GetOAuthV2Info>
```

Sample policy:

```
<GetOAuthV2Info name="GetClientAttributes">
  <ClientId>{client_id}</ClientId>
</GetOAuthV2Info>
```

Variables set on success:

- `oauthv2accesstoken.{policy_name}.access_token`
- `oauthv2accesstoken.{policy_name}.scope`
- `oauthv2accesstoken.{policy_name}.refresh_token`
- `oauthv2accesstoken.{policy_name}.accesstoken.{custom_attribute_name}`
- `oauthv2accesstoken.{policy_name}.developer.id`
- `oauthv2accesstoken.{policy_name}.developer.app.name`
- `oauthv2accesstoken.{policy_name}.expires_in`

`oauthv2accesstoken.{policy_name}.status`

Sample policy:

```
   <GetOAuthV2Info name="GetTokenAttributes">
      <AccessToken ref="{variable_name}"/>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Sample policy:

```
      </GetOAuthV2Info>
```

Sample policy:

```
   <GetOAuthV2Info name="GetTokenAttributes">
   <AccessToken>{access_token}</AccessToken>
</GetOAuthV2Info>
```

Get authorization code attributes policy

Sample policy:

```
   <GetOAuthV2Info name="GetAuthCodeAttributes">
    <AuthorizationCode ref="{variable_name}"/>
   </GetOAuthV2Info>
```

Sample policy:

```
   <GetOAuthV2Info name="GetAuthCodeAttributes">
       <AuthorizationCode>{authorization_code}</AuthorizationCode>
</GetOAuthV2Info>
```

Get refresh token attributes policy

Sample policy:

```
   <GetOAuthV2Info name="GetTokenAttributes">
     <RefreshToken ref="{variable_name}"/>
   </GetOAuthV2Info>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Sample policy:

```
   <GetOAuthV2Info name="GetTokenAttributes">
       <RefreshToken>{refresh_token}</RefreshToken>
</GetOAuthV2Info>
```

7.8.2 OAuth 1.0a Flow Variables

The flow variables defined in this table are populated when the respective OAuth policies are executed, and hence are available to other policies or applications executing in the API proxy flow.

Generate request token policy

Sample policy:

```
   <OAuthV1 name="GenerateRequestToken">
     <Operation>GenerateRequestToken</Operation>
   </OAuthV1>
```

Variables set on success:

- oauth_token
- oauth_token_secret
- oauth_callback_confirmed
- oauth_response
- oauth_consumer_key
- oauth_consumer_secret

Generate access token policy

Sample policy:

```
   <OAuthV1 name="GenerateAccessToken">
     <Operation>GenerateAccessToken</Operation>
   </OAuthV1>
```

Variables set on success:

- oauth_token
- oauth_token_secret
- oauth_response
- oauth_consumer_key
- oauth_consumer_secret

Access token verification policy

Sample policy:

```
   <OAuthV1 name="VerifyAccessToken">
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <Operation>VerifyAccessToken</Operation>
  </OAuthV1>
```

Variables set on success:

- oauth_token
- oauth_token_secret
- oauth_response
- oauth_consumer_key
- oauth_consumer_secret

Verify API key policy

Sample policy:

```
  <GetOAuthV1Info name="VerifyApiKey">
    <OAuthConfig>{config_name}</OAuthConfig>*
    <APIKey ref="{variable_name}" />
  </GetOAuthV1Info>
```

** Optional*

Variables set on success:

- oauth_consumer_key
- oauth_consumer_secret

Verify consumer policy

Sample policy:

```
  <GetOAuthV1Info name="VerifyConsumer">
    <OAuthConfig>{config_name}</OAuthConfig>*
    <ConsumerKey ref="{variable_name}" />
  </GetOAuthV1Info>
```

** Optional*

Variables set on success:

- oauth_consumer_key
- oauth_consumer_secret

Verify token policy

Sample policy:

```
  <GetOAuthV1Info name="VerifyToken">
    <OAuthConfig>{config_name}</OAuthConfig>*
    <RequestToken ref="{variable_name}" />
  </GetOAuthV1Info>
```

* Optional

Variables set on success:

- oauth_token
- oauth_token_secret

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.9 Variables Reference

SAP API Management defines a set of variables that are available out-of-the-box. This section defines all the variables that are "pre-defined" by API Platform.

> **Note**
>
> If you're new to flow variables or have general questions about them, see Introduction to flow variables. For information about how variables are used in conditional flows, see Flow variables and conditions.

7.9.1 About variable scope

Flow variables have scope. By scope, we mean the point in the proxy flow life cycle when a variable is first instantiated. For example, if you have a policy attached to the ProxyEndpoint request segment that policy will not be able to access any variables that are scoped to the TargetEndpoint request segment. The reason for this is that the TargetEndpoint request segment of the flow has not executed yet, so the API proxy hasn't had a chance to populate variables in that scope.

> **Note**
>
> It is important to note that if you try to access a flow variable that is out of scope, you will receive a NULL value. If you try to set a flow variable before it is in scope, the proxy does nothing; it does not generate an error and does not set the variable.

For more information see Understanding Flow Variable Scope.

7.9.2 Request message variables

Using the variables that apply to the request message, policies may access message components including the header, the query parameters, and form parameters, the source IP address, the HTTP message body, and so on.

The proxy receives an incoming request message, and then applies to it a series of policies, based on conditions evaluated on the request, which can modify or transform the request. In the normal flow, the proxy then sends the transformed request to the target.

Policies can examine request variables, then transform or reject the request based on the content of those variables. Policies transform the request by setting the appropriate variables, for example variables corresponding to the request headers.

Request message variables	Scope	Type	Permission	Description
client.cn	Proxy request	String	Read	The common name specified in the SSL certificate presented by the client app.
client.country	Proxy request	String	Read	The country in the SSL certificate presented by the client app.
client.email.address	Proxy request	String	Read	The email address in the SSL certificate presented by the client app.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Request message variables	Scope	Type	Permission	Description
client.host	Proxy request	String	Read	The HTTP host associated with the request received by the ProxyEndpoint.
client.ip	Proxy request	String	Read	The originating IP address of the request received by the ProxyEndpoint.
client.organization	Proxy request	String	Read	The organization in the SSL certificate presented by the client. (Not necessarily equivalent to the organization on SAP API Management.)
client.organization.unit	Proxy request	String	Read	The organizational unit in the SSL certificate presented by the client.
client.locality	Proxy request	String	Read	The locality (City) in the SSL certificate presented by the client.
client.port	Proxy request	Integer	Read	The HTTP port associated with the originating client request to ProxyEndpoint.
client.received.end.time	Proxy request	String	Read	The time, expressed in string form, at which the proxy finished receiving the request from the originating client at the ProxyEndpoint. For example: Wed, 21 Aug 2013 19:16:47 UTC This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.
client.received.end.timestamp	Proxy request	Long	Read	The timestamp value specifying when the proxy finished receiving the request from the originating client at the ProxyEndpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.
client.received.start.time	Proxy request	String	Read	The time, expressed in string form, at which the proxy began receiving the request from the originating client at the ProxyEndpoint. For example: Wed, 21 Aug 2013 19:16:47 UTC This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Request message variables	Scope	Type	Permission	Description
				UTC' corresponds to the timestamp value of 1377112607413.
client.received.start.timestamp	Proxy request	Long	Read	The timestamp value specifying when the proxy began receiving the request from the originating client at the ProxyEndpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.
client.sent.end.time	Proxy response	String	Read	The time, expressed in string form, when the ProxyEndpoint finished returning the response to the originating client app. For example: Wed, 21 Aug 2013 19:16:47 UTC This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.
client.sent.end.timestamp	Proxy response	Long	Read	The timestamp value specifying when the ProxyEndpoint finished returning the response to the originating client app. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.
client.sent.start.time	Proxy response	String	Read	The time, expressed in string form, when the ProxyEndpoint started returning the response to the originating client app. For example: Wed, 21 Aug 2013 19:16:47 UTC This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.
client.sent.start.timestamp	Proxy response	Long	Read	The timestamp value specifying when the ProxyEndpoint started returning the response to the originating client app. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Request message variables	Scope	Type	Permission	Description
client.state	Proxy request	String	Read	The state in the SSL certificate presented by the client.
client.scheme	Proxy request	String	Read	Returns http or https depending on the transport used by client app to send the request message.
client.ssl.enabled	Proxy request	String	Read	Returns true or false, depending on whether the ProxyEndpoint is configured for SSL.
message.path	Proxy request	String	Read/Write	The complete request message path in URL excluding any query parameters.
message.queryparam.{queryparam_name}	Proxy request	String	Read	Returns the specified message query parameter.
message.queryparam.{queryparam_name}.values.count	Proxy request	Integer	Read	The total count of a specified query parameter associated with the request sent to the ProxyEndpoint from the client app.
message.queryparams.count	Proxy request	Integer	Read	The total count of all query parameters associated with the request sent to the ProxyEndpoint from the client app.
message.queryparams.names	Proxy request	Collection	Read	A list of all query parameter names associated with the request sent to the ProxyEndpoint from the client app.
message.querystring	Proxy request	String	Read	A string containing all query parameter names and values associated with the request sent to the ProxyEndpoint from the client app. For example, where the request is: `http://api.apifactory.com/inventors?name=nikola&surname=tesla` this variable returns: name=nikola&surname=tesla
message.uri	Proxy request	String	Read	The complete URI path (following the domain URL) including query parameters. For example, where the request is: `http://api.apifactory.com/inventors?name=nikola&surname=tesla`

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Request message variables	Scope	Type	Permission	Description
				this variable returns: inventors?name=nikola&surname=tesla
message.verb	Proxy request	String	Read	The HTTP verb (GET, PUT, POST, DELETE, etc.) associated with the request
message.version	Proxy request	String	Read/Write	The HTTP version associated with the request sent to the ProxyEndpoint from the client app.
messageid	Proxy request	String	Read	Holds the globally unique ID for the request. This ID allows requests received at the router to be tracked after they are sent to the message processor. This ID is logged in API Platform error logs to correlate the messageId with the errors.
proxy.basepath	Proxy request	String	Read/Write	The value of API proxy basepath that is sent from the client and received at the ProxyEndpoint. The basepath is defined as the path component that uniquely identifies the API proxy. The public-facing URL of an API proxy is comprised of your organization name, the environment where the proxy is deployed, the basepath, the basepath suffix, and any query parameters. For example, in the following request: `http://myorg-test.mycompany.net/v2/weatherapi/forecastrss?w=12797282` The basepath is `/v2/weatherapi`.
proxy.client.ip	Proxy request	String	Read	The IP address associated with the HTTP request from the requesting client app.
proxy.pathsuffix	Proxy request	String	Read	The value of API proxy basepath suffix that is sent from the client and received at the ProxyEndpoint.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Request message variables	Scope	Type	Permission	Description
				The basepath is defined as the path component that uniquely identifies the API proxy. The public-facing URL of an API proxy is comprised of your organization name, the environment where the proxy is deployed, the basepath, the basepath suffix, and any query parameters. For example, in the following request: `http://myorg-test.mycompany.net/v2/weatherapi/forecastrss?w=12797282` The basepath suffix is /forecastrss
proxy.url	Proxy request	String	Read	Gets the complete URL associated with the proxy request received by the ProxyEndpoint, including any query parameters present.
request	Proxy request	Message	Read	The complete request, including any payload present.
request.content	Proxy request	String	Read/Write	Gets or sets the payload of the request message.
request.formparam.{formparam_name}	Proxy request	String	Read/Write	Gets or sets the value of the specified form parameter in the request sent from the client app.
request.formparam.{formparam_name}.values.count	Proxy request	Integer	Read	Count of all values for the specified form parameter associated with the request.
request.formparams.count	Proxy request	Integer	Read	Count of all form parameters associated with the request sent from the client app.
request.formparams.names	Proxy request	Collection	Read	A list of all form parameter names associated with the request.
request.formstring	Proxy request	String	Read	The complete formparam in the request sent from the client app. For example: name=test&type=first&group=A
request.header.{header_name}	Proxy request	String	Read/Write	Gets or sets the value of a particular header found in the request.
request.header.{header_name}.values	Proxy request	Collection	Read	All the values of a particular header in the request

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Request message variables	Scope	Type	Permission	Description
request.header.{header_name}.values.count	Proxy request	Integer	Read	Count of all the values of a particular header in the request.
request.headers.count	Proxy request	Integer	Read	Count of all the headers in the request.
request.headers.names	Proxy request	Collection	Read	Names of all the headers in the request.
request.path	Proxy request	String	Read/Write	The complete request path in the request URL excluding query parameters. This value is the same as the basepath and basepath suffix combined. For example: If the basepath of the request is /123, and the basepath suffix is /456/hello, then request.path returns /123/456/test.
request.queryparam.{queryparam_name}	Proxy request	String	Read/Write	The value of a particular query parameter found in the request.
request.queryparam.{queryparam_name}.values.count	Proxy request	Integer	Read	The count of all the values of a particular query parameter in the request.
request.queryparams.count	Proxy request	Integer	Read	The count of all the query parameters in the request.
request.queryparams.names	Proxy request	Collection	Read	The names of all the query parameters in the request.
request.querystring	Proxy request	String	Read	The complete list of query parameters in the request sent from the client app. For example, if the request is `http://host.com/123?name=first&surname=second&place=address` then this variable returns name=first&surname=second&place=address
request.transportid	Proxy request	String	Read	ID of the request as type TransportMessage which is a contextual object
request.transport.message	Proxy request	Transport Message	Read	Request of type TransportMessage which is a contexual object
request.uri	Proxy request	String	Read/Write	The value of the request.path variable including query parameters.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Request message variables	Scope	Type	Permission	Description
request.verb	Proxy request	String	Read/Write	The HTTP verb used for the request. For example: GET, PUT, DELETE
request.version	Proxy request	String	Read	Gets the HTTP version of the request.
response.formstring	Target request	String	Read	The complete list of form parameters in the request. For example: name=test&type=first&group=A
route.target	Target request	String	Read	The TargetEndpoint name. For example: default.
servicecallout.{policy-name}.expectedcn	Proxy request	String	Read/Write	The expected Common Name of the TargetEndpoint as referred to in a ServiceCallout policy. This is meaningful only when the TargetEndpoint refers to an SSL endpoint.
servicecallout.{policy-name}.target.url	Proxy request	String	Read/Write	The TargetEndpoint URL for a particular ServiceCallout policy. See also, Call services or APIs using ServiceCallout.
target.basepath	Target request	String	Read	Returns basepath of TargetEndpoint
target.basepath.with.query	Target request	String	Read	Returns basepath with queryparam of TargetEndpoint.
target.copy.pathsuffix	Target request	Boolean	Read/Write	When true, request forwarded from ProxyEndpoint to TargetEndpoint retains path sufffix (the URI path fragment following the URI defined in the ProxyEndpoint base path.)
target.copy.queryparams	Target request	Boolean	Read/Write	When true, request forwarded from ProxyEndpoint to TargetEndpoint retains query parameters.
target.cn	Target request	String	Read	The Common Name of the TargetEndpoint. This is meaningful only when the TargetEndpoint refers to an SSL endpoint.
target.expectedcn	Proxy request	String	Read/Write	The expected Common Name of the TargetEndpoint. This is meaningful only when the TargetEndpoint refers to an SSL endpoint.
target.host	Proxy request	String	Read	Target host to which message is reaching from TargetEndpoint

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Request message variables	Scope	Type	Permission	Description
target.name	Target request	String	Read	Target to which message is reaching from targetendpoint
target.ip	Proxy request	String	Read	Target ip to which message is reaching from targetendpoint
target.port	Proxy request	Integer	Read	Target source port which is connected to TargetEndpoint port for recieving request.
targetserver.name	Target request	String	Read	Target server name to which message is reaching from targetendpoint
target.scheme	Target request	String	Read/Write	Returns http or https depending on the request message
target.sent.end.time	Target request	String	Read	The time, expressed in string form, at which the proxy stopped sending the request to the URL specified in the TargetEndpoint. For example: Wed, 21 Aug 2013 19:16:47 UTC This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.
target.sent.end.timestamp	Target request	Long	Read	The timestamp value specifying when the proxy finished sending the request to the URL specified in the TargetEndpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.
target.sent.start.time	Target request	String	Read	The time, expressed in string form, at which the proxy began sending the request to the URL specified in the TargetEndpoint. For example: Wed, 21 Aug 2013 19:16:47 UTC This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.
target.sent.start.timestamp	Target request	Long	Read	The timestamp value specifying when the proxy started sending the request

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Request message variables	Scope	Type	Permission	Description
				to the URL specified in the TargetEndpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.
target.ssl.enabled	Proxy request	Boolean	Read	Whether TargetEndpoint is running on SSL
target.url	Target request	String	Read/Write	When read, returns the complete URL set in the TargetEndpoint, including any query parameters. When written, sets the complete URL for the TargetEndpoint. Note: Use a JavaScript policy to set this variable.
variable.expectedcn	Proxy request	String	Read/Write	Variable exposed for the common name if it's running on SSL
virtualhost.aliases	Proxy request	String_array	Read	Host aliases of the virtual host that is hit during a particular request
virtualhost.name	Proxy request	String	Read	Name of the virtual host that serves the originating client request
virtualhost.ssl.enabled	Proxy request	Boolean	Read	Returns true if SSL is enabled in the virtual host configuration

7.9.3 Response Message Variables

Using the variables that apply to the response message, policies may access message components including the header, the query parameters, and form parameters, the source IP address, the HTTP message body, and so on.

The proxy receives a response message, and then applies to it a series of policies, based on conditions evaluated on the response, which can modify or transform the response.

Policies can examine response variables, then transform or reject the request based on the content of those variables. Policies transform the response by setting the appropriate variables, for example variables corresponding to the response headers.

Response message variable	Scope	Type	Permission	Description
`client.sent.end.time`	Proxy response	String	Read	The time, expressed in string form, at which the proxy finished sending the response from the ProxyEndpoint to the client. For example: Wed, 21 Aug 2013 19:16:47 UTC

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Response message variable	Scope	Type	Permission	Description
				This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.
`client.sent.end.timestamp`	Proxy response	Long	Read	The timestamp value specifying when the proxy finished sending the response to the client from the ProxyEndpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.
`client.sent.start.time`	Proxy response	String	Read	The time, expressed in string form, at which the proxy began sending the response from the ProxyEndpoint to the client. For example: Wed, 21 Aug 2013 19:16:47 UTC This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413
`client.sent.start.timestamp`	Proxy response	Long	Read	The timestamp value specifying when the proxy began sending the response to the client from the ProxyEndpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.
`loadbalancing.failedservers`	Target response	String_ array	Read	List of failed TargetServers during load balancing at TargetEndpoint

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Response message variable	Scope	Type	Permission	Description
`loadbalancing.isfallback`	Target response	Boolean	Read	Return true if fallback is enabled for the TargetServer invoked during load balancing at TargetEndpoint
`loadbalancing.targetserver`	Target response	String	Read	TargetServer invoked during load balancing at TargetEndpoint
`message.reason.phrase`	Target response	String	Read	ReasonPhrase of the response message from target
`message.status.code`	Target response	Integer	Read	HTTP status code of the response message from target
`response`	Target response	Message	Read/Write	Complete response message returned by target
`response.content`	Target response	String	Read/Write	Payload content of the response message returned by the target
`response.formparam.{formparam_name}`	Target response	String	Read/Write	The value of a form parameter in the response
`response.formparam.{formparam_name}.values.count`	Target response	Integer	Read	Count of all the values of the specified form parameter in response
`response.formparams.count`	Target response	Integer	Read	Count of all form prameters in the response
`response.formparams.names`	Target response	Collection	Read	The names of all the form parameters in the response
`response.header.{header_name}`	Target response	String	Read/Write	Gets or sets the value of a specified HTTP header in the response
`response.header.{header_name}.values`	Target response	Collection	Read	All the values of a specified HTTP header in response
`response.header.{header_name}.values.count`	Target response	Integer	Read	Count of all the values of the specified HTTP header in response
`response.headers.count`	Target response	Integer	Read	Count of all the headers in the response
`response.headers.names`	Target response	Collection	Read	The names of all the headers in the response

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Response message variable	Scope	Type	Permission	Description
`response.reason.phrase`	Target response	String	Read/Write	The response reason phrase for a particular request
`response.status.code`	Target response	Integer	Read/Write	The response code returned for a particular request
`response.transport.message`	Target response	String	Read	Response of type TransportMessage which is a contexual object
`target.country`	Target response	String	Read	Country of the SSL certificate presented by the target server
`target.email.address`	Target response	String	Read	Email address of the SSL certificate presented by the target server
`target.organization`	Target response	String	Read	Organization of the SSL certificate presented by the target server
`target.organization.unit`	Target response	String	Read	Organization unit of the SSL certificate presented by the target server
`target.locality`	Target response	String	Read	Locality (city) of the SSL certificate presented by the target server
`target.received.end.time`	Target response	String	Read	The time, expressed in string form, at which the TargetEndpoint started receiving the response from the target. For example: Wed, 21 Aug 2013 19:16:47 UTC This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.
`target.received.end.timestamp`	Target response	Long	Read	The timestamp value specifying when the TargetEndpoint finished receiving the response from the target. This value is a 64-bit (long) integer containing the number of milliseconds

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Response message variable	Scope	Type	Permission	Description
				elapsed since midnight, on January 1, 1970 UTC.
`target.received.start.time`	Target response	String	Read	The time, expressed in string form, at which the TargetEndpoint finished receiving the response from the target. For example: Wed, 21 Aug 2013 19:16:47 UTC This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.
`target.received.start.timestamp`	Target response	Long	Read	The timestamp value specifying when the TargetEndpoint started receiving the response from the target. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.
`target.state`	Target response	String	Read	State of the SSL certificate presented by the target server

7.9.4 Message variables

Message variables are contextual: they refer to different message types (request, response, or error) depending on the point within the API Proxy Flow in which they are called. The supported message variables are listed below.

`Message variables`	Scope	Type	Permission	Description
`error`	Error	Message	Read / Write	Error of type Message which is a contextual object in the error flow
`error.content`	Error	String	Read / Write	Content of the error
`error.message`	Error	String	Read	Message associated with an error, whose value is available only

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Message variables	Scope	Type	Permission	Description
				before the error Flow is executed
`error.transport.message`	Error	Transport_Message	Read	Any error of type TransportMessage
`error.state`	Error	Integer	Read	State in the Flow where an error occured
`message`	Always	Message	Read/Write	A contextual object, with the same value as `request in` the request Flow or as `response in` the response Flow or as `error` in the Error flow.
`message.content`	Always	String	Read/Write	Content of the request or response message
`message.formparam.{formparam_name}`	Always	String	Read/Write	Value of the specified form parameter
`message.formparam.{formparam_name}.values`	Always	Collection	Read	All values of the specified form parameter in the message
`message.formparam.{formparam_name}.values.count`	Always	Integer	Read	Count of the values of the specified form parameters in the message
`message.formparams.count`	Always	Integer	Read	Count of all form parameters in the message
`message.formparams.names`	Always	Collection	Read	Value of all form parameters in the message
`message.formstring`	Always	String	Read	Value of form string in the message
`message.header.{header_name}`	Always	String	Read/Write	Gets or sets the value of the specified HTTP header in the message
`message.header.{header_name}.values`	Always	Collection	Read	All values of the specified HTTP

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Message variables	Scope	Type	Permission	Description
				header name in the message
`message.header.{header_name}.values.count`	Always	Integer	Read	Count of the values of the specified HTTP header name in the message
`message.headers.count`	Always	Integer	Read	Count of all HTTP headers in the message
`message.headers.names`	Always	Integer	Read	Value of all HTTP headers in the message
`messagelogging.{policy-name}.failed`	Always	Boolean	Read	Failure flag for the referenced Message logging policy
`messagelogging.failed`	Always	Boolean	Read	Failure flag for Message logging policy
`message.queryparam.{queryparam_name}.values`	Always	Integer	Read	Value of the specified query parameter in the message
`message.transport.message`	Always	Transport_Message	Read	Message of type `TransportMessage` which is a contextual object

7.9.5 Path Variables

Path variables are predefined variables that can be used to retrieve different parts of the request path.

Variables	Path	Description
Request variables	`request.uri`	The HTTP request path, which includes a path and a query string separated by a question mark (?)
	`request.path`	The HTTP request path without the query string
	`request.querystring`	The portion of the HTTP request path after the question mark (?)
Application variables	application.basepath	The deployment base path (specified during API deployment)

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Variables	Path	Description
Proxy Variables	proxy.basepath	The base path as configured in the proxy XML file
	proxy.pathprefix	The portion of the request path after the proxy basepath
Target variables	target.url	The URL configured in the target XML file or the dynamic target URL (if target.url is set during the message flow)
	target.basepath	The path (without host or port) in the URL configured in the target XML file or the dynamic target URL (if target.url is set during the message flow)
	target.basepath.with.query	The path along with query params (without host or port) in the URL configured in the target XML file or the dynamic target URL (if target.url is set during the message flow)

Example

Consider the following deployment:

1. Deployment path: /dev/
2. Proxy base path: /twitter/
3. Virtual host port: 8090
4. Virtual host alias: mytwitter.com
5. Target URL: http://api.twitter.com/1/

   Request from client - The client makes the request:

   ```
   GET /dev/twitter/help/test.json
   curl http://{host}:8090/dev/twitter/help/test.json -H 'host: mytwitter.com'
   ```

   The path is visualized as :

   ```
   {VERB} {proxy.basepath}/{proxy.pathsuffix}
   ```

   Request is sent to target as:

   ```
   GET /1/help/test.json
   equivalent to:  curl http://api.twitter.com/1/help/test.json
   ```

   The path is visualized as:

   ```
   {VERB} {target.basepath}/{proxy.pathsuffix}?{query params}
   ```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Usage in flows

Paths are usually used to classify requests as REST resources. This is achieved using conditional flows in a path match condition. Consider the following two requests based on the example above, which are two resources:

```
GET /dev/twitter/help/test.json
GET /dev/twitter/statuses/public_timeline.json
```

You can enforce different policies on these resources. The proxy.pathsuffix variable policy is recommended in conditions.

> **Note**
>
> You cannot use request.path because it includes the deployment path, which can change. The same bundle should work with different deployment paths.

7.9.6 Configuration Variables

Configuration variables represent the configuration settings for an organization on SAP API Management. The supported configuration variables are listed below:

Configuration variables	Scope	Type	Permission	Description
`organization.name`	Always	String	Read	Name of the organization
`environment.name`	Always	String	Read	Name of the environment
`apiproxy.name`	Always	String	Read	Name of the apiproxy
`apiproxy.revision`	Always	String	Read	The revision number of an API proxy
`is.error`	Always	Boolean	Read	Error flag
`proxy.basepath`	Always	Boolean	Read	The base path (URI) configured for the ProxyEndpoint
`proxy.name`	Always	String	Read	The name attribute configured for the ProxyEndpoint
`proxy.pathsuffix`	Always	String	Read	The request URI path fragment following the URI base path URI fragment

7.9.7 System Variables

System variables contain system-related information. A system variable is prefixed with "`_system.`" wherever it is referenced. The part of the variable that appears after the prefix specifies the function in the system, such as host name, date, time, and interfaces. The supported system variables are listed below.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

System variable	Scope	Type	Permission	Description
system.timestamp	Always	Long	Read	The timestamp value specifying when the request is received from the client at the ProxyEndpoint. This value is a 64-bit (long) integer containing the number of milliseconds elapsed since midnight, on January 1, 1970 UTC.
system.time	Always	String	Read	The time, expressed in string form, at which the proxy received a request from a client at the ProxyEndpoint. For example: Wed, 21 Aug 2013 19:16:47 UTC This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.
system.time.year	Always	Integer	Read	The year portion of the system.time variable.
system.time.month	Always	Integer	Read	The month portion of the system.time variable.
system.time.day	Always	Integer	Read	The day of month portion of the system.time variable.
system.time.dayofweek	Always	Integer	Read	The day of the week portion of the system.time variable.
system.time.hour	Always	Integer	Read	The hour portion of the system.time variable.
system.time.minute	Always	Integer	Read	The minute portion of the system.time variable.
system.time.second	Always	Integer	Read	The second portion of the system.time variable.
system.time.millisecond	Always	Integer	Read	The millisecond portion of the system.time variable.
system.time.zone	Always	String	Read	Timezone of the system
system.interface.{interface_name}	Always	String	Read	IP Address of the system
system.pod.name	Always	String	Read	The name of the pod where the proxy is running

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

System variable	Scope	Type	Permission	Description
system.region.name	Always	String	Read	The name of the data center region where the proxy is running
system.uuid	Always	String	Read	The UUID of the message processor handling the proxy
router.uuid	Always	String	Read	The UUID of the router handling the proxy

7.9.8 SSL / TLS Variables

The following variables are populated when an API proxy executes a transaction over inbound SSL/TLS. For the variables to be populated the property propagate.additional.ssl.headers must set to true in the ProxyEndpoint configuration.

Example

```
<ProxyEndpoint name="myProxy">
  <HTTPProxyConnection>
  <BasePath>/v1/weather</BasePath>
  <VirtualHost>secure</VirtualHost>
  <Properties>
    <Property name="propagate.additional.ssl.headers">true</Property>
  </Properties>
</HTTPProxyConnection>
```

The variables are designed to be at parity with the Apache web server SSL settings.

SSL variable	Scope	Permission	Description
ssl.protocol	Proxy request	Read	See Apache Module mod_ssl.
ssl.session.id	Proxy request	Read	See Apache Module mod_ssl.
ssl.cipher	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.m.version	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.m.serial	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.i.dn	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.i.dn.cn	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.i.dn.c	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.i.dn.o	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.i.dn.l	Proxy request	Read	See Apache Module mod_ssl.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

SSL variable	Scope	Permission	Description
ssl.server.s.dn	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.s.dn.cn	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.s.dn.c	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.s.dn.o	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.s.dn.l	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.v.start	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.v.end	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.v.remain	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.a.sig	Proxy request	Read	See Apache Module mod_ssl.
ssl.server.a.key	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.m.version	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.m.serial	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.i.dn	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.i.dn.cn	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.i.dn.o	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.i.dn.c	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.i.dn.l	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.s.dn	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.s.dn.cn	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.s.dn.o	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.s.dn.c	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.s.dn.l	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.v.start	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.v.end	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.v.remain	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.a.sig	Proxy request	Read	See Apache Module mod_ssl.
ssl.client.a.key	Proxy request	Read	See Apache Module mod_ssl.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

7.9.9 Policy Variables

Each Policy type defines a set of policy-specific variables. For details, see each Policy type under the Policy reference overview.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8 Policy Reference

8.1 Policy Reference Overview

By adding an API to the API platform, you gain the ability to modify and monitor its behavior using out-of-the-box policies. The out-of-the-box policies let you add sophisticated features to control traffic, enhance performance, enforce security, and increase the utility of your APIs, without coding or changing backend services. Extension policies enable you to add custom logic in the form of JavaScript, Python and XSLT to an API, tailoring it to meet business requirements.

Following are the categories of policies that SAP API Management provides.

Traffic management policies

- Traffic management policies let you configure cache, control traffic quotas and spikes, set concurrent rate limits, and so on.

Security policies

- Security policies let you control access to your APIs with OAuth, API key validation, and other threat protection features.

Mediation policies

- Mediation policies let you perform message transformation, parsing, and validation, as well as raise faults and alerts.

Extension policies

- Extension policies let you provide custom policy functionality beyond what is provided by SAP API Management, with support for such features as a service callout, message data collection, and calling JavaScript, and Python behavior you have created.

8.2 Traffic Management Policies

- Optimize performance using cache
- Rate limit API traffic using Quota
- Reduce latency using ResponseCache
- Shield APIs using SpikeArrest
- Reset quota counter using ResetQuota
- Throttle Backend Connections Using ConcurrentRatelimit

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.2.1 Optimize Performance Using Cache

Using the following policies, your proxy can store and retrieve the cached data at runtime:

- PopulateCache policy to add data to the cache.
- LookupCache policy to access cached data.
- InvalidateCache policy to flush the cache.

In the following example, an OAuth access token is written to the cache using a PopulateCache policy. The OAuth token is retrieved for subsequent requests by a LookupCache policy. Once the access token expires, JavaScript is used to retrieve a new access token, which is in turn cached by the PopulateCache policy.

Populate the Cache

Use the PopulateCache policy to write data to the cache. This example writes an OAuth access token to the cache. For policy reference information, see PopulateCache policy.

```
<PopulateCache name="token-cache">
    <!-- The cache to write to. -->
    <CacheResource>mycache</CacheResource>
    <!-- The source of the data, a variable containing the value. -->
    <Source>twitter-translate.apiAccessToken</Source>
    <!-- An enumeration representing a prefix for namespace scope. -->
    <Scope>Exclusive</Scope>
    <!-- A unique pointer to the data. Use this later to retrieve it. -->
    <CacheKey>
        <KeyFragment>apiAccessToken</KeyFragment>
        <KeyFragment>request.queryparam.client_id</KeyFragment>
    </CacheKey>
    <!-- Fragments constructing the cache entry should expire after 600 seconds. -->
    <ExpirySettings>
        <TimeoutInSec>600</TimeoutInSec>
    </ExpirySettings>
</PopulateCache>
```

Variables can be populated by policies or by code. The Source variable in this example is populated by the following JavaScript call:

```
context.setVariable('twittertranslate.apiAccessToken',getAccessToken());
```

For more on cache keys, see Working with cache keys.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Lookup Cached Data

You can retrieve cached values with the LookupCache policy. The following LookupCache policy reads a value from mycache and writes the value to the variable `twitter-translate. apiAccessToken`. For policy reference information, see LookupCache policy.

```
<LookupCache name="token-cache">
    <!-- The cache to read from. -->
    <CacheResource>mycache</CacheResource>
    <!-- Where to assign the retrieved value - here, a variable. -->
    <AssignTo>twitter-translate.apiAccessToken</AssignTo>
    <!-- An enumeration representing a prefix for namespace scope. -->
    <Scope>Exclusive</Scope>
    <!-- Fragments constructing the unique pointer used when the data was put into the
cache. -->
    <CacheKey>
        <KeyFragment>apiAccessToken</KeyFragment>
        <KeyFragment>request.queryparam.client_id</KeyFragment>
    </CacheKey>
</LookupCache>
```

Invalidate the Cache

The cache can be invalidated explicitly by specifying an HTTP header. When a request that contains the specified HTTP header is received, the cache will be flushed. For policy reference information, see InvalidateCache policy.

```
<InvalidateCache name="InvalidateMyCache">
    <!-- The cache to invalidate. -->
    <CacheResource>test-cache</CacheResource>
    <!-- An enumeration representing a prefix for namespace scope. -->
    <Scope>Exclusive</Scope>
    <!-- Fragments constructing the unique pointer used when
        the data was put into the cache. -->
    <CacheKey>
        <KeyFragment>apiAccessToken</KeyFragment>
        <KeyFragment>request.queryparam.client_id</KeyFragment>
        <!-- Flush the cache when this header is received. -->
        <KeyFragment ref="request.header.invalidate_cache" />
    </CacheKey>
    <PurgeChildEntries>true</PurgeChildEntries>
</InvalidateCache>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.2.1.1 Cache Policy Variables and Error Codes

This topic lists variables and error codes related to caching policies. For more about caching, see Optimize performance using cache.

Variables

Flow variables can be used to configure dynamic runtime behavior for policies and flows, based on HTTP headers or message content, or the context available in the Flow. For more information about flow variables, see Variables reference.

The following predefined Flow variables are available after you customize the behavior of the cache you define in a LookupCache policy. For more about the policy, see LookupCache policy.

Variables	Type	Permission	Description
lookupcache.<policy-name>.cachename	String	Read-only	Returns the cache name used in the policy.
lookupcache.<policy-name>.cachekey	String	Read-only	Returns the key used.
lookupcache.<policy-name>.cachehit	Boolean	Read-only	True if the policy execution is successful.
lookupcache.<policy-name>.assignto	String	Read-only	Returns the variable to which cache is assigned.

Error Codes

Error Code	Message
CacheNotFound Cache	{0}, not found in organization : {1}
InvalidCacheResourceReference	Invalid cache resource reference {0} in Stepdefinition {1}. Context {2}
EntryCannotBeCached Entry	{0} can not be cached. Only serializable entries are cached
CannotDeleteStepDefinition	Step definition {0} should be detached before deletion
InvalidTimeout	CacheLookupTimeoutInSeconds {0} value should be greater than zero

8.2.1.2 InvalidateCache Policy

Configures how the cached values should be purged from the cache.

Element reference

The element reference describes the elements and attributes of the InvalidateCache policy.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<InvalidateCache async="false" continueOnError="false" enabled="true" name="policy-
name">
    <DisplayName>Policy Name</DisplayName>
    <FaultRules>rules_for_fault_handling</FaultRules>
    <Properties>property_elements</Properties>
    <CacheKey>
        <Prefix>prefix_string</Prefix>
        <KeyFragment ref="variable_reference"/>
        <KeyFragment>fragment_string</KeyFragment>
    </CacheKey>
    <!-- Omit this element if you're using the included shared cache. -->
    <CacheResource>cache_to_use</CacheResource>
    <Scope>scope_enumeration</Scope>
    <CacheContext>
        <APIProxyName>application_that_added_the_entry</APIProxyName>
        <ProxyName>proxy_for_which_data_was_cached</ProxyName>
        <TargetName>endpoint_for_which_data_was_cached</TargetName>
    </CacheContext>
    <PurgeChildEntries>true_to_purge_all_child_entries</PurgeChildEntries>
</InvalidateCache>
```

Your organization comes with a default cache. For more about the underlying data store, see Why Use a Cache?. For more about configuring caches, see Manage caches for an environment.

<InvalidateCache> Attributes

Field Name	Description	Default	Required
name	The unique name of the policy. Use this name to reference the policy within a Step definition element. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the `<DisplayName>` element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Yes
continueOnError	Set to `false` to return an error when a policy fails. This is expected behavior for most policies.	false	Optional

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name	Description	Default	Required
	Set to `true` to have flow execution continue even after a policy fails.		
enabled	Set to `true` to enforce the policy. Set to `false` to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to `true`, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to `true` improves API proxy performance. However, overusing async can hurt performance with too much thread switching. To use asynchronous behavior in API proxies, see Java Callout Policy.	false	Optional

8.2.1.2.1 <DisplayName> Element

You can use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

```
<DisplayName>Policy Display Name</DisplayName>
```

Default: N/A
If you omit this element, the the value of the policy's name attribute is used.

Presence: Optional

Type: String

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.2.1.2.2 <CacheContext> / <APIProxyName> Element

This element specifies the name of the application that added the cache entry.

```
<APIProxyName>application_that_added_the_entry</APIProxyName>
```

Attributes

Attribute	Description	Default	Presence	Type
ref	Variable with the application name.	N/A	Optional	String

8.2.1.2.3 <CacheContext> Element

This element specifies how to construct a cache key when a `Prefix` element value is not specified, or to clear cache entries added by another API proxy.

```
<CacheContext>
  <APIProxyName ref="variable_name">application_that_added_the_entry</ApplicationName>
  <TargetName ref="variable_name">endpoint_for_which_data_was_cached</TargetName>
  <ProxyName ref="variable_name">proxy_for_which_data_was_cached</ProxyName>
</CacheContext>
```

Used to construct the CacheKey. Values for APIProxyName, ProxyName, and TargetName are mandatory when a CacheKey prefix (that is, a custom prefix) is not used to clear cache entries added by another API proxy.

8.2.1.2.4 <CacheKey> Element

Configures a unique pointer to a piece of data stored in the cache.

```
<CacheKey>
    <Prefix>string</Prefix>
    <KeyFragment ref="variable_name" />
    <KeyFragment>literal_string</KeyFragment>
</CacheKey>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

<CacheKey> constructs the name of each piece of data stored in the cache. The cache key you define in the PopulateCache policy is the name that you use in the LookupCache policy to get the specific entry that you need from the cache.

At runtime, <KeyFragment> values are prepended with either the <Scope> element value or <Prefix> value. For example, the following results in a cache key of UserToken__apiAccessToken__<value_of_client_id>:

```
<CacheKey>
    <Prefix>UserToken</Prefix>
    <KeyFragment>apiAccessToken</KeyFragment>
    <KeyFragment ref="request.queryparam.client_id" />
</CacheKey>
```

You use the <CacheKey> element in conjunction with Prefix and Scope. For more information, see Working with cache keys.

8.2.1.2.5 <CacheResource> Element

This element specifies the cache where messages should be stored. A default cache is available.

Omit this element completely if this policy (and your corresponding PopulateCache and LookupCache policies) is using the included shared cache.

```
<CacheResource>cache_to_use</CacheResource>
```

For more about configuring caches, see Manage caches for an environment.

8.2.1.2.6 <CacheKey> / <KeyFragment> Element

This element specifies a value that should be included in the cache key, creating a namespace for matching requests to cached responses.

```
<KeyFragment ref="variable_name"/>
<KeyFragment>literal_string</KeyFragment>
```

Default:	N/A
Presence:	Optional
Type:	N/A

This can be a key (a static name that you provide) or a value (a dynamic entry set by referencing a variable). All specified fragments combined (plus the prefix) are concatenated to create the cache key.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<KeyFragment>apiAccessToken</KeyFragment>
<KeyFragment ref="request.queryparam.client_id" />
```

You use the <KeyFragment> element in conjunction with <Prefix> and <Scope>. For more information, see Working with Cache Keys

Attributes

Attribute	Type	Default	Required	Description
ref	string		No	The variable from which to get the value. Should not be used if this element contains a literal value.

<CacheKey>/<KeyFragment> elementThis element pecifies a value to use as a cache key prefix.

<Prefix>prefix_string</Prefix>

You can use this value instead of **<Scope>** when you want to specify your own value rather than a **<Scope>** - enumerated value. If defined, **<Prefix>** prepends the cache key value for entries written to the cache. A**<Prefix>** element value overrides a **<Scope>** element value.

You use the **<Prefix>** element in conjunction with **<CacheKey>** and **<Scope>**. For more information, see Working with cache keys.

8.2.1.2.7 <CacheKey>/<Prefix> Element

This element specifies a value to use as a cache key prefix.

```
<Prefix>prefix_string</Prefix>
```

Default: N/A

Presence: Optional

Type: String

You can use this value instead of <Scope> when you want to specify your own value rather than a <Scope> - enumerated value. If defined, <Prefix> prepends the cache key value for entries written to the cache. A <Prefix> element value overrides a <Scope> element value.

You use the <Prefix> element in conjunction with <CacheKey> and <Scope>. For more information, see Working with cache keys.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.2.1.2.8 <CacheContext> / <ProxyName> Element

This element specifies the name of the proxy for which the data was cached.

```
<ProxyName>proxy_for_which_data_was_cached</ProxyName>
```

Default: N/A

Presence: Optional

Type: String

Attributes

Attribute	Description	Default	Presence	Type
ref	The variable from which to get the value. Should not be used if this element contains a literal value.	N/A	Optional	String

8.2.1.2.9 <PurgeChildEntries> Element

`true` to purge cache entries that share the prefix value set by the `<Prefix>` element configured for this policy. Values in other parts of the cache key, such as in `<KeyFragment>` elements, are not considered.

Note that the `<Prefix>` element must be specified. If the prefix is not set, setting true for `<PurgeChildEntries>` could result in purging all entries in the cache.

Invalidating all of the cache entries of the same prefix value can be a useful way to purge multiple related entries at once.

```
<PurgeChildEntries>true_to_purge_child_entries</PurgeChildEntries>
```

Default: false

Presence: Optional

Type: Boolean

8.2.1.2.10 <Scope> Element

Enumeration used to construct a prefix for a cache key when a Prefix element is not provided in the CacheKey element.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<Scope>scope_enumeration</Scope>
```

The Scope setting determines a cache key that is prepended according to the Scope value. For example, a cache key would take the following form when the scope is set to Exclusive: `orgName__envName__apiProxyName__deployedRevisionNumber__proxy|TargetName__ [ serialize dCacheKey ]`.

If a Prefix element is present in CacheKey, it supersedes a Scope element value. Valid values include the enumerations below.

You use the Scope element in conjunction with CacheKey and Prefix. For more information, see Working with cache keys.

Scope Values

Global	Cache key is shared across all API proxies deployed in the environment. Cache key is prepended in the form `orgName __ envName __`. If you define a **<CacheKey>** entry with the **<KeyFragment>** apiAccessToken and a **<Global>** scope, each entry is stored as orgName__envName__apiAccessToken, followed by the serialized value of the access token. For an API proxy deployed in an environment called 'test' in an organization called 'apifactory', access tokens would be stored under the following cache key: apifactory__test__apiAccessToken.
Application	API proxy name is used as the prefix. Cache key is prepended in the form `orgName__envName__apiProxyName`.
Proxy	ProxyEndpoint configuration is used as the prefix. Cache key is prepended in the form `orgName__envName__apiProxyName__deployedRevisionNumber__proxyEndpointName`.
Target	TargetEndpoint configuration is used as the prefix. Cache key prepended in the form `orgName__envName__apiProxyName__deployedRevisionNumber__targetEndpointName`.
Exclusive	Default. This is the most specific, and therefore presents minimal risk of namespace collisions within a given cache. Prefix is one of two forms: · If the policy is attached to the `ProxyEndpoint` flow, prefix is of the form ApiProxyName_ProxyEndpointName. · If the policy is attached at `TargetEndpoint`, prefix is of the form ApiProxyName_TargetName. Cache key prepended in the form · `orgName__envName__apiProxyName__deployedRevisionNumber__proxyNameITarget Name` For example, the full string might look like this: apifactory__test__weatherapi__16__default__apiAccessToken .

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.2.1.2.11 <CacheContext> / <TargetName> Element

This element specifies the name of the target endpoint for which the data was cached.

```
<TargetName>endpoint_for_which_data_was_cached</TargetName>
```

Default: N/A

Presence: Optional

Type: String

Attributes

Attribute	Description	Default	Presence	Type
ref	The variable from which to get the value. Should not be used if this element contains a literal value.	N/A	Optional	String

8.2.1.3 LookupCache Policy

Configures how cached values should be retrieved at runtime.

Element reference

The element reference describes the elements and attributes of the LookupCache policy.

```
<LookupCache async="false" continueOnError="false" enabled="true" name="Lookup-Cache-1">
    <DisplayName>Lookup Cache 1</DisplayName>
    <FaultRules/>
    <Properties/>
    <CacheKey>
        <Prefix/>
        <KeyFragment ref=""/>
    </CacheKey>
    <!-- Omit this element if you're using the included shared cache. -->
    <CacheResource/>
    <Scope>Exclusive</Scope>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <AssignTo>flowVar</AssignTo>
</LookupCache>
```

At runtime, the LookupCache policy retrieves a value from the cache, assigning the value to the variable you specify with the AssignTo element. It looks for the value based on a cache key created, through configuration that combines the CacheKey and Scope elements. In other words, to retrieve a particular value added to the cache by a PopulateCache policy, your LookupCache policy must have cache key-related elements configured in the same way.

Your organization comes with a default cache. For more about the underlying data store, see Why Use a Cache? For more about configuring caches, see Manage caches for an environment.

<LookupCache> Attributes

Field Name	Description	Default	Required
name	The unique name of the policy. Use this name to reference the policy within a Step definition element. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the `<DisplayName>` element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Yes
continueOnError	Set to `false` to return an error when a policy fails. This is expected behavior for most policies. Set to `true` to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to `true` to enforce the policy. Set to `false` to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to `true`, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to `true` improves API	false	Optional

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name	Description	Default	Required
	proxy performance. However, overusing async can hurt performance with too much thread switching. To use asynchronous behavior in API proxies, see Java Callout Policy.		

8.2.1.3.1 <DisplayName> Element

You can use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

```
<DisplayName>Policy Display Name</DisplayName>
```

Default:	N/A If you omit this element, the the value of the policy's name attribute is used.
Presence:	Optional
Type:	String

8.2.1.3.2 <AssignTo> Element

This element specifies the variable where the cache entry is assigned after it has been retrieved from the cache. The variable must be writable.

```
<AssignTo>variable_to_receive_cached_value</AssignTo>
```

Default:	N/A
Presence:	Required
Type:	String

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.2.1.3.3 <CacheKey> Element

Configures a unique pointer to a piece of data stored in the cache.

```
<CacheKey>
    <Prefix>string</Prefix>
    <KeyFragment ref="variable_name" />
    <KeyFragment>literal_string</KeyFragment>
</CacheKey>
```

<CacheKey> constructs the name of each piece of data stored in the cache.

CacheKey constructs the name of each piece of data stored in the cache. The cache key you define in the `PopulateCache` policy is the name that you use in the `LookupCache` policy to get the specific entry that you need from the cache.

At runtime, `<KeyFragment>` values are prepended with either the Scope element value or Prefix value. For example, the following results in a cache key of `UserToken__apiAccessToken__<value_of_client_id>`:

```
<CacheKey>
    <Prefix>UserToken</Prefix>
    <KeyFragment>apiAccessToken</KeyFragment>
    <KeyFragment ref="request.queryparam.client_id" />
</CacheKey>
```

You use the <CacheKey> element in conjunction with `<Prefix>` and <Scope>. For more information, see Working with cache keys.

8.2.1.3.4 <CacheResource> Element

This element specifies the cache where messages should be stored.

Omit this element completely if this policy (and your corresponding LookupCache and InvalidateCache policies) is using the included shared cache.

```
<CacheResource>cache_to_use</CacheResource>
```

Default:	N/A
Presence:	Optional
Type:	String

For more about configuring caches, see Creating and Editing an Environment Cache.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.2.1.3.5 <CacheKey> / <KeyFragment> Element

This element specifies a value that should be included in the cache key, creating a namespace for matching requests to cached responses.

```
<KeyFragment ref="variable_name"/>
<KeyFragment>literal_string</KeyFragment>
```

Default:	N/A
Presence:	Optional
Type:	N/A

This can be a key (a static name that you provide) or a value (a dynamic entry set by referencing a variable). All specified fragments combined (plus the prefix) are concatenated to create the cache key.

```
<KeyFragment>apiAccessToken</KeyFragment>
<KeyFragment ref="request.queryparam.client_id" />
```

You use the <KeyFragment> element in conjunction with <Prefix> and <Scope>. For more information, see Working with Cache Keys

Attributes

Attribute	Type	Default	Required	Description
ref	string		No	The variable from which to get the value. Should not be used if this element contains a literal value.

<CacheKey>/<KeyFragment> elementThis element pecifies a value to use as a cache key prefix.

<Prefix>prefix_string</Prefix>

You can use this value instead of **<Scope>** when you want to specify your own value rather than a **<Scope>** - enumerated value. If defined, **<Prefix>** prepends the cache key value for entries written to the cache. A**<Prefix>** element value overrides a **<Scope>** element value.

You use the **<Prefix>** element in conjunction with **<CacheKey>** and **<Scope>**. For more information, see Working with cache keys.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.2.1.3.6 <CacheKey> / <Prefix> Element

This element specifies a value to use as a cache key prefix.

```
<Prefix>prefix_string</Prefix>
```

Default:	N/A
Presence:	Optional
Type:	String

You can use this value instead of <Scope> when you want to specify your own value rather than a <Scope> - enumerated value. If defined, <Prefix> prepends the cache key value for entries written to the cache. A <Prefix> element value overrides a <Scope> element value.

You use the <Prefix> element in conjunction with <CacheKey> and <Scope>. For more information, see Working with cache keys.

8.2.1.3.7 <Scope> Element

Enumeration used to construct a prefix for a cache key when a <Prefix> element is not provided in the <CacheKey> element.

```
<Scope>scope_enumeration</Scope>
```

The <Scope> setting determines a cache key that is prepended according to the <Scope> value. For example, a cache key would take the following form when the scope is set to Exclusive:

```
orgName__envName__apiProxyName__deployedRevisionNumber__proxy|TargetName__ [ serialize
dCacheKey ].
```

If a **<Prefix>** element is present in **<CacheKey>**, it supercedes a **<Scope>** element value. Valid values include the enumerations below.

You use the **<Scope>** element in conjunction with **<CacheKey>** and **<Prefix>.** For more information, see Working with cache keys.

Scope values

Global	Cache key is shared across all API proxies deployed in the environment. Cache key is prepended in the form `orgName __ envName __`. If you define a **<CacheKey>** entry with the **<KeyFragment>** apiAccessToken and a **<Global>** scope, each entry is stored as orgName__envName__apiAccessToken, followed by the

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

	serialized value of the access token. For an API proxy deployed in an environment called 'test' in an organization called 'apifactory', access tokens would be stored under the following cache key: apifactory__test__apiAccessToken.
Application	API proxy name is used as the prefix. Cache key is prepended in the form `orgName__envName__apiProxyName`.
Proxy	ProxyEndpoint configuration is used as the prefix. Cache key is prepended in the form `orgName__envName__apiProxyName__deployedRevisionNumber__proxyEndpointName`.
Target	TargetEndpoint configuration is used as the prefix. Cache key prepended in the form `orgName__envName__apiProxyName__deployedRevisionNumber__targetEndpointName`.
Exclusive	Default. This is the most specific, and therefore presents minimal risk of namespace collisions within a given cache. Prefix is one of two forms: · If the policy is attached to the `ProxyEndpoint` flow, prefix is of the form ApiProxyName_ProxyEndpointName. · If the policy is attached at `TargetEndpoint`, prefix is of the form ApiProxyName_TargetName. Cache key prepended in the form · `orgName__envName__apiProxyName__deployedRevisionNumber__proxyNameITarget Name` For example, the full string might look like this: apifactory__test__weatherapi__16__default__apiAccessToken .

8.2.1.3.8 LookupCache-Specific Variables

Flow variables can be used to configure dynamic runtime behavior for policies and flows, based on HTTP headers or message content, or the context available in the Flow. For more information about flow variables, see Variables reference.

The following predefined Flow variables are available after you customize the behavior of the cache you define in a LookupCache policy.

Variables	Type	Permission	Description
lookupcache.<policy-name>.cachename	String	Read-only	Returns the cache name used in the policy.
lookupcache.<policy-name>.cachekey	String	Read-only	Returns the key used.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Variables	Type	Permission	Description
lookupcache.<policy-name>.cachehit	Boolean	Read-only	True if the policy execution is successful.
lookupcache.<policy-name>.assignto	String	Read-only	Returns the variable to which cache is assigned.

8.2.1.3.9 Error Codes

Error Code	Message
`CacheNotFound Cache`	`{0}, not found in organization : {1}`
`InvalidCacheResourceReference`	`Invalid cache resource reference {0} in Step definition {1}. Context {2}`
`EntryCannotBeCached Entry`	`{0} can not be cached. Only serializable entries are cached.`
`CannotDeleteStepDefinition`	`Step definition {0} should be detached before deletion`
`InvalidTimeout`	`CacheLookupTimeoutInSeconds {0} value should be greater than zero`

8.2.1.4 PopulateCache Policy

Configures how cached values should be written at runtime.

Element reference

The element reference describes the elements and attributes of the PopulateCache policy.

```
<PopulateCache async="false" continueOnError="false" enabled="true" name="Populate-Cache-1">
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <DisplayName>Populate Cache 1</DisplayName>
    <FaultRules/>
    <Properties/>
    <CacheKey>
        <Prefix/>
        <KeyFragment ref=""/>
    </CacheKey>
    <!-- Omit this element if you're using the included shared cache. -->
    <CacheResource/>
    <Scope>Exclusive</Scope>
    <ExpirySettings>
        <TimeoutInSec>300</TimeoutInSec>
    </ExpirySettings>
    <Source>flowVar</Source>
</PopulateCache>
```

At runtime, the **<PopulateCache>** policy writes data from the variable you specified in the **<Source>** element to the cache you specified in the **<CacheResource>** element. You can use the **<CacheKey>**, **<Scope>**, and **<Prefix>** elements to specify a key that you can use from the **<LookupCache>** policy to retrieve the value. Use the **<ExpirySettings>** element to configure when the cached value should expire.

Your organization comes with a default cache. For more about the underlying data store, see Why Use a Cache? For more about configuring caches, see Manage caches for an environment.

<PopulateCache> Attributes

Field Name	Description	Default	Required
name	The unique name of the policy. Use this name to reference the policy within a Step definition element. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the `<DisplayName>` element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Yes
continueOnError	Set to `false` to return an error when a policy fails. This is expected behavior for most policies.	false	Optional

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name	Description	Default	Required
	Set to `true` to have flow execution continue even after a policy fails.		
enabled	Set to `true` to enforce the policy. Set to `false` to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to `true`, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to `true` improves API proxy performance. However, overusing async can hurt performance with too much thread switching. To use asynchronous behavior in API proxies, see Java Callout Policy.	false	Optional

8.2.1.4.1 <DisplayName> Element

You can use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

```
<DisplayName>Policy Display Name</DisplayName>
```

Default: N/A
If you omit this element, the the value of the policy's name attribute is used.

Presence: Optional

Type: String

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.2.1.4.2 <CacheKey> Element

Configures a unique pointer to a piece of data stored in the cache.

```
<CacheKey>
    <Prefix>string</Prefix>
    <KeyFragment ref="variable_name" />
    <KeyFragment>literal_string</KeyFragment>
</CacheKey>
<CacheKey>
```

Default:	N/A
Presence:	Required
Type:	N/A

<CacheKey> constructs the name of each piece of data stored in the cache.

Constructs the name of each piece of data stored in the cache. The cache key you define in the <PopulateCache> policy is the name that you use in the <LookupCache> policy to get the specific entry that you need from the cache.

At runtime, <KeyFragment> values are prepended with either the <Scope> element value or <Prefix> value. For example, the following results in a cache key of UserToken__apiAccessToken__<value_of_client_id>:

```
<CacheKey>
    <Prefix>UserToken</Prefix>
    <KeyFragment>apiAccessToken</KeyFragment>
    <KeyFragment ref="request.queryparam.client_id" />
</CacheKey>
```

You use the <CacheKey> element in conjunction with <Prefix> and <Scope>. For more information, see Working with cache keys.

8.2.1.4.3 <CacheResource> Element

This element specifies the cache where messages should be stored.

Omit this element completely if this policy (and your corresponding LookupCache and InvalidateCache policies) is using the included shared cache.

```
<CacheResource>cache_to_use</CacheResource>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Default: N/A

Presence: Optional

Type: String

For more about configuring caches, see Creating and Editing an Environment Cache.

8.2.1.4.4 <ExpirySettings> / <ExpiryDate> Element

This element pecifies the date on which a cache entry should expire. Use the form mm-dd-yyyy.

```
<ExpirySettings>
    <ExpiryDate ref="{date_variable}">expiration_date</ExpiryDate>
</ExpirySettings>
```

Default: N/A

Presence: Optional

Type: String

Attributes

Attribute	Description	Default	Presence	Type
ref	The variable from which to get the value. Should not be used if this element contains a literal value.	N/A	Optional	String

8.2.1.4.5 <ExpirySettings> Element

This element specifies when a cache entry should expire.

```
<ExpirySettings>
  <TimeOfDay ref="time_variable">expiration_time</TimeOfDay>
  <TimeoutInSec ref="duration_variable">seconds_until_expiration</TimeoutInSec>
  <ExpiryDate ref="date_variable">expiration_date</ExpiryDate>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
</ExpirySettings>
```

Default:	N/A
Presence:	Required
Type:	N/A

8.2.1.4.6 <CacheKey>/<KeyFragment> Element

This element specifies a value that should be included in the cache key, creating a namespace for matching requests to cached responses.

```
<KeyFragment ref="variable_name"/>
<KeyFragment>literal_string</KeyFragment>
```

Default:	N/A
Presence:	Optional
Type:	N/A

This can be a key (a static name that you provide) or a value (a dynamic entry set by referencing a variable). All specified fragments combined (plus the prefix) are concatenated to create the cache key.

```
<KeyFragment>apiAccessToken</KeyFragment>
<KeyFragment ref="request.queryparam.client_id" />
```

You use the <KeyFragment> element in conjunction with <Prefix> and <Scope>. For more information, see Working with Cache Keys

Attributes

Attribute	Type	Default	Required	Description
ref	string		No	The variable from which to get the value. Should not be used if this element contains a literal value.

<CacheKey>/<KeyFragment> elementThis element pecifies a value to use as a cache key prefix.

```
<Prefix>prefix_string</Prefix>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

You can use this value instead of **<Scope>** when you want to specify your own value rather than a **<Scope>** - enumerated value. If defined, **<Prefix>** prepends the cache key value for entries written to the cache. A**<Prefix>** element value overrides a **<Scope>** element value.

You use the **<Prefix>** element in conjunction with **<CacheKey>** and **<Scope>**. For more information, see Working with cache keys.

8.2.1.4.7 <CacheKey> / <Prefix> Element

This element specifies a value to use as a cache key prefix.

```
<Prefix>prefix_string</Prefix>
```

Default:	N/A
Presence:	Optional
Type:	String

You can use this value instead of <Scope> when you want to specify your own value rather than a <Scope> - enumerated value. If defined, <Prefix> prepends the cache key value for entries written to the cache. A <Prefix> element value overrides a <Scope> element value.

You use the <Prefix> element in conjunction with <CacheKey> and <Scope>. For more information, see Working with cache keys.

8.2.1.4.8 <Scope> Element

Enumeration used to construct a prefix for a cache key when a <Prefix> element is not provided in the <CacheKey> element.

```
<Scope>scope_enumeration</Scope>
```

The <Scope> setting determines a cache key that is prepended according to the <Scope> value. For example, a cache key would take the following form when the scope is set to Exclusive:

```
orgName__envName__apiProxyName__deployedRevisionNumber__proxy|TargetName__ [ serialize
dCacheKey ].
```

If a **<Prefix>** element is present in **<CacheKey>**, it supercedes a **<Scope>** element value. Valid values include the enumerations below.

You use the **<Scope>** element in conjunction with **<CacheKey>** and **<Prefix>.** For more information, see Working with cache keys.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Scope values

Global	Cache key is shared across all API proxies deployed in the environment. Cache key is prepended in the form `orgName __ envName __`. If you define a **<CacheKey>** entry with the **<KeyFragment>** apiAccessToken and a **<Global>** scope, each entry is stored as orgName__envName__apiAccessToken, followed by the serialized value of the access token. For an API proxy deployed in an environment called 'test' in an organization called 'apifactory', access tokens would be stored under the following cache key: apifactory__test__apiAccessToken.
Application	API proxy name is used as the prefix. Cache key is prepended in the form `orgName__envName__apiProxyName`.
Proxy	ProxyEndpoint configuration is used as the prefix. Cache key is prepended in the form `orgName__envName__apiProxyName__deployedRevisionNumber__proxyEndpointName`.
Target	TargetEndpoint configuration is used as the prefix. Cache key prepended in the form `orgName__envName__apiProxyName__deployedRevisionNumber__targetEndpointName`.
Exclusive	Default. This is the most specific, and therefore presents minimal risk of namespace collisions within a given cache. Prefix is one of two forms: · If the policy is attached to the `ProxyEndpoint` flow, prefix is of the form ApiProxyName_ProxyEndpointName. · If the policy is attached at `TargetEndpoint`, prefix is of the form ApiProxyName_TargetName. Cache key prepended in the form `orgName__envName__apiProxyName__deployedRevisionNumber__proxyNameITargetName` For example, the full string might look like this: apifactory__test__weatherapi__16__default__apiAccessToken .

8.2.1.4.9 <Source> Element

This element specifies the variable whose value should be written to the cache.

```
<Source>source_variable</Source>
```

Default: N/A

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Presence: Required

Type: String

8.2.1.4.10 <ExpirySettings> / <TimeOfDay> Element

The time of day at which a cache entry should expire. Use the form `HH:mm:ss`. When present, this element's sibling, `<TimeoutInSec>`, overrides `<TimeOfDay>`.

Enter time of day in the format `HH:mm:ss`, where HH represents the hour on a 24-hour clock. For example, 14:30:00 for 2:30 in the afternoon.

For the time of day, the default locale and time zone will vary depending on where the code is running (which isn't knowable when you configure the policy). For information on configuring your locale, see Creating and Editing an Environment Cache.

```
<ExpirySettings>
    <TimeOfDay ref="time_variable">expiration_time</TimeOfDay>
</ExpirySettings>
```

Default: N/A

Presence: Optional

Type: String

Attributes

Attribute	Description	Default	Presence	Type
ref	Variable with the expiration time value.	N/A	Optional	String

8.2.1.4.11 <ExpirySettings> / <TimeoutInSec> Element

The number of seconds after which a cache entry should expire.

```
<ExpirySettings>
    <TimeoutInSec ref="duration_variable">seconds_until_expiration</TimeoutInSec>
</ExpirySettings>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Default: N/A

Presence: Optional

Type: Integer

Attributes

Attribute	Description	Default	Presence	Type
ref	Variable with the timeout value.	N/A	Optional	String

8.2.1.5 Why Use a Cache?

You can improve performance by caching data your proxy uses. For a simple example, see Optimize performance using cache.

You can access the cache with policies. For reference information, see PopulateCache policy, LookupCache policy, and InvalidateCache policy.

You might want a dedicated custom cache to:

- Reduce latency: A request satisfied from the cache gets the representation and displays it in a shorter time. The server is more responsive to requests satisfied from the cache, which is closer to the client than the origin server.
- Reduce network traffic: Representations are reused, reducing the impact of processing duplicate or redundant requests. Using a cache also reduces the amount of bandwidth you use.
- Persist data across transactions: You can store session data for reuse across HTTP transactions.
- Support security: You may need "scope" access to the contents of a cache so it can only be accessed in a particular environment or by a specific API proxy.

8.2.1.5.1 About the Underlying Data Store

Data is cached in the underlying persistence store, a Cassandra (NoSQL) database. Your organization is provisioned with a default cache that stores entries in this data store. You don't need to know about the persistence store implementation to use caching.

The default cache works well for basic use cases, but you can also create dedicated custom caches for specific applications. For more about creating a cache, see Manage caches for an environment.

You can create multiple caches in your organization. To support data segregation, a cache's scope is the environment in which the cache is created. For example, API proxies running in a "test" environment cannot

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

access data in a cache running in "prod". Once you create a cache, you can use cache policies to cache any serializable data.

Recommendation

For guidance on which type of cache or persistence to use in different situations, see Persistence.

8.2.1.6 Working with Cache Keys

When using a cache from a proxy, you can ensure the uniqueness of cached value keys by configuring cache keys. A cache key, along with other values you can configure, gives you a reliable way to get out the data that you put in.

The values of configuration elements -- CacheKey / KeyFragment, Scope, and Prefix -- are concatenated to create a unique identifier that is associated with the value you put into the cache with the PopulateCache policy. You use this same configuration to retrieve the value with the LookupCache policy.

For reference, about these elements, see PopulateCache Policy, LookupCache policy, InvalidateCache Policy, Reduce latency using ResponseCache

With the following cache policy configuration elements, you can create a namespace in which values are unique:

Cache Configuration Element	Description
`CacheKey > KeyFragment`	Use `<CacheKey> <KeyFragment>` elements combine to specify a unique identifier for cache entries. KeyFragment values can be static literals or set from variables.
`<Scope>` or `<Prefix>`	Use the `<Scope>` or `<Prefix>` elements to namespace cache keys. Scope enumerates a list of predefined values. The Prefix element overrides Scope with a value of your own choosing.

These values are concatenated in the following form, with `Scope>` or `<Prefix>` values separated from KeyFragment values by double-underscores. Multiple KeyFragment values are also separated by double underscores.

```
scope | prefix__keyfragment[__keyfragment]
```

8.2.1.6.1 Using CacheKey

The `<CacheKey>` element specifies a unique identifier for cache entries created from a PopulateCache policy. When you use a LookupCache policy to retrieve the cached values, you use a `<CacheKey>` element defined in the same way to get the same entry.

The `<CacheKey>` element can include multiple `<KeyFragment>` elements. Values of `<Key Fragment>` elements are concatenated with two underscores between them.

The following configuration creates a cache key of `hello__world`:

```
<CacheKey>
    <KeyFragment>hello</KeyFragment>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <KeyFragment>world</KeyFragment>
<CacheKey>
```

You can also use a variable value in a cache key by referencing the variable in a KeyFragment element.

```
<KeyFragment ref="variable_name"/>
```

For example, to make the CacheKey value unique to the Content-Type of the request message, you do as follows:

```
<KeyFragment ref="request.header.Content-Type"/>
```

This defines a KeyFragment as the value of the context variable `request.header.Content-Type`. In the configuration below, the `request.header.Content-Type` variable has the value `application/json`.

```
<CacheKey>
  <KeyFragment>apiAccessToken</KeyFragment>
  <KeyFragment ref="request.header.Content-Type" />
  <KeyFragment>bar</KeyFragment>
</CacheKey>
```

This result a cache key of `<Scope>__apiAccessToken__application/json__bar`.

Cache keys derived from query parameters

Using variables such as `request.queryparam.<queryparam_name>` and `request.querystring`, you can configure a cache key so that the key includes parts of a request's query string. For example, the following URL uses two query parameters -- param1 and param2 -- that you can use in your cache key:

http://myaccount.<host:port>/mydata?param1=value1&param2=value2

Your `<CacheKey>` element can incorporate these values with a configuration such as the following:

```
<CacheKey>
    <KeyFragment ref="request.queryparam.param1" />
    <KeyFragment ref="request.queryparam.param2" />
<CacheKey>
```

At runtime, the cache key would include the param values concatenated, as in the following:

```
other_key_parts__value1__value2
```

Keep in mind that when you use variables to insert values from parameters, the values will be concatenated in the order suggested by the `<KeyFragment>` element order. Also, note that SAP API Management will use only those values that you specifically reference with `<KeyFragment>` elements. If your request query parameter lists vary, the variations won't be accounted for in the cache key.

An alternative is to use the request.querystring variable, which inserts the entire string of parameters literally as part of the cache key. Keep in mind that while this method accounts for all of the parameters, if the order of parameters varies from one request to the next then the key will be different. In other words,

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

`param1=value1&param2=value2` and `param2=value2&param1=value1` don't result in the same cache key value.

8.2.1.6.2 Using Scope and Prefix

The <Scope> and <Prefix> elements provide a way to augment the key with a namespace prefix. The values they represent are prepended to your cache key.

The <Scope> element is used by default. It is an enumeration whose value ranges from broad to narrow, with the narrowest as the default. This default value is used unless you specify another value or specify a <Prefix> element value. You can override the Scope value by using a <Prefix> element, and so specify a custom value for namespacing.

For example, the <Scope> value "Global" -- the broadest scope -- represents the organization and environment name. So if your proxy is deployed in an organization called 'mycompany' and an environment named 'prod', the resulting preprended value will be the following:

Configuration	Result
<Scope>Global</Scope>	mycompany__prod__.

If you're using the Global scope with the cache key defined above, the result is as follows:

Configuration	Result
<Scope>Global</Scope> **<CacheKey>** **<KeyFragment>hello</KeyFragment>** **<KeyFragment>world</KeyFragment>** **<CacheKey>**	mycompany__prod__hello__world.

As described in LookupCache policy, the scope can be configured for increasing specificity from Global to Exclusive. An Exclusive Scope is the most specific, and therefore represents minimal risk of namespace collisions within a given cache. Each cache entry with an Exclusive scope is prefixed in the following form:

orgName__envName__apiProxyName__deployedRevisionNumber__proxy|TargetName__[serializedCacheKey]

For example, a cache key created from using the Exclusive value for Scope would look like the following:

apifactory__test__weatherapi__16__default__apiAccessToken

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.2.2 Rate Limit API Traffic Using Quota

- Quota types
- Identifying apps
- Dynamic Quota settings
- Time notation
- Configuring a Quota policy
- Policy-specific variables
- Policy-specific error codes

A Quota is an allotment of request messages that an app is allowed to submit to an API over the course of an hour, a day, a week, or a month. To support Quota enforcement, SAP API Management maintains counters that tally the number of requests received from individual apps. This capability enables API providers to enforce limits on the number of API calls made by apps over an interval of time. Using Quota policies you can, for example, limit apps to 1 request per minute, or to 10,000 requests per month.

When an app reaches its Quota limit, subsequent API calls are rejected; SAP API Management returns an error message for every request that exceeds the Quota. The Quota policy provides the ability to reset Quotas on-the-fly, after an app has exceeded its limit.

Example

For example, if a Quota is defined as 10,000 messages per month, rate-limiting begins after the 10,000th message. It doesn't matter whether 10,000 messages were counted on the first day or the last day of that period; the app is blocked from making additional requests until the Quota counter automatically resets at the end of the specified time interval, or until the Quota is explicitly reset using Reset quota counter using ResetQuota.

Note

A variation on Quota called SpikeArrest prevents traffic spikes (or bursts) that can be caused by a sudden increase in usage, buggy clients, or malicious attacks. For more information on SpikeArrest, see Shield APIs using SpikeArrest.

Note

The name attribute for this policy is restricted to these characters: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.

8.2.2.1 Quota Types

Three types of Quota are supported. You define the Quota type as an attribute of the Quota policy.

Example

```
<Quota name="MyQuota" type="calendar">
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Note

The following Quota types are supported:

- `calendar`: In the calendar Quota, you define an explicit start time. The Quota counter for each app is refreshed based on the Interval and TimeUnit that you set.
- For example, the following Quota of type calendar begins counting at 10 am on February 18, 2014, and will refresh every 5 hours.

```
<Quota name="QuotaPolicy" type="calendar">
  <Identifier ref="request.header.clientId"/>
  <StartTime>2014-02-18 10:00:00</StartTime>
  <Interval>5</Interval>
  <TimeUnit>hour</TimeUnit>
  <Allow count="99"/>
  <MessageWeight ref="request.header.weight"/>
  <Distributed>true</Distributed>
  <Synchronous>true</Synchronous>
</Quota>
```

- `rollingwindow`: A "rolling window" counter advances by the time interval that you specify. You don't specify a StartTime; instead, the StartTime for the counter is the time when the first message is received from the client plus the interval that you define. A counter is kept for each client ID (consumer key). Thus, the counter will reset to zero when the Interval you define has passed. This enables you to configure a Quota in which an app is indefinitely allowed 1000 requests every 24 hours.
- `flexi`: Flexible Quota enforcement causes the counter to begin when the first request message is received from an app. Under flexible Quota enforcement, StartTime is dynamic; every app has its own StartTimebased on the time when the first request is received. This enables you to provide Quotas that support one week, one month, or 6 months access to your API, customized for each app.
- Default: A Quota policy that does not explicitly define a type is a default Quota configuration. In the default configuration, quota enforcement type is calendar without a StartTime configured. The following is an example of a default Quota policy.

```
<Quota name="MyQuota">
  <Identifier ref="request.header.clientId"/>
  <Interval>1</Interval>
  <TimeUnit>hour</TimeUnit>
  <Allow count="10000"/>
</Quota>
```

- When you use a default, Quota enforcement takes place based on calendar time intervals--always based on the timestamp when the first request message is received by the API proxy. In the above example, imagine that the first message is received at `2013-07-08 07:35:28`. The StartTime for the Quota count will be `2013-07-08 07:00:00` and the EndTime will be `2013-07-08 08:00:00` (1 hour from the StartTime). TheStartTime will always be the calendar or clock StartTime of the corresponding interval, such as hour, day, week, or month.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.2.2.2 Identifying Apps

Quotas are counted for individual apps. For this to work, each app requires a unique identifier that it presents with each request. The identifier can be customized. It can be any HTTP header, query parameter, form parameter, or message content that is unique to each consumer app.

You specify the token that will identify each app by defining a reference to a variable in the Identifier element. The Identifier most commonly used to uniquely identify apps is the client_id. The client_id is another name for the API key, or consumer key that is generated for an app when it is registered in an organization on SAP API Management. You can use this identifier if you have enabled API key or OAuth authorization policies for your API.

<Identifier ref="request.header.client_id"/>

In some circumstances, Quota settings must be retrieved where no client_id is available, such as when no security policy is in place. In those situations, you can use the AccessEntity policy type to retrieve the appropriate API product settings.

8.2.2.3 Dynamic Quota Settings

Quota settings can be static or dynamic. The following examples demonstrate static and dynamic Quota configurations.

Sample static Quota Configuration:

In a simple, static Quota, you set a count, a time interval, and a time unit. The following static Quota configuration limits requesting apps to 10,000 requests per month:

```
<Quota name="QuotaPolicy">
    <Allow count="10000"/>
    <Interval>1</Interval>
    <TimeUnit>month</TimeUnit>
</Quota>
```

In the example above the count is 10,000 client requests, over the course of 1 month.

API Product-based Dynamic Quota Configuration:

Dynamic Quota configurations are more useful than static Quotas. Dynamic Quotas enable you to configure a single Quota policy that enforces different Quota settings for different apps; based on the identity of the requesting app. (Another term for Quota settings in this context is "Service Plan". The dynamic Quota checks the apps' "Service Plan" and then enforces those settings.)

Dynamic Quota settings are populated at runtime by resolving an app identifier to an API product. The Identifier can be a field in the request that is unique to each app. For API proxies where OAuth is enforced, you can use client_id as the Identifier, as demonstrated in the sample policy below.

```
<Quota name="CheckQuota">
  <Interval ref="apiproduct.developer.quota.interval"/>
  <TimeUnit ref="apiproduct.developer.quota.timeunit"/>
  <Allow countRef="apiproduct.developer.quota.limit"/>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  <Identifier ref="client_id"/>
</Quota>
```

If an API Key is used to validate an inbound request, then, assuming that the policy is named "VerifyApiKey", then the dynamic Quota policy should be configured as follows:

```
<Quota name="ratelimiting.developer.quotas">
  <Identifier ref="request.queryparam.apikey" />
  <Interval ref="verifyapikey.VerifyApiKey.apiproduct.developer.quota.interval" />
  <TimeUnit ref="verifyapikey.VerifyApiKey.apiproduct.developer.quota.timeunit" />
  <Allow countRef="verifyapikey.VerifyApiKey.apiproduct.developer.quota.limit" />
</Quota>
```

The example above uses the variable `client_id` to identify the requesting app. This works as long as the request message contains a `client_id` (associated with an OAuth-enabled request).

Note that the value for `verifyapikey.VerifyApiKey` will vary based on the name of the OAuthV2 policy of type VerifyApiKey that is used to check the API key. If your policy is called, for example, KeyValidationPolicy, then the variable name will be `verifyapikey.KeyValidationPolicy`.

8.2.2.4 Time Notation

All Quotas are set to the Coordinated Universal Time (UTC) time zone.

Quota time notation follows the international standard date notation defined in International Standard ISO 8601.

Dates are defined as year, month, and day, in the following format: `YYYY-MM-DD`.

For example, `2013-02-04` represents February 4, 2013.

Time of day is defined as hours, minutes, and second, in the following format: hours: minutes: seconds.

For example, 23:59:59 represents the time one second before midnight.

Note that two notations, `00:00` and `24:00`, are available to distinguish the two midnights that can be associated with one date. Therefore:

`2013-02-04 24:00:00` is the same date and time as `2013-02-05 00:00:00`

00:00 is usually the preferred notation.

8.2.2.5 Configuring a Quota Policy

Configure Quota policy using the following elements.

XML File Name	Description
StartTime (Optional)	Use only for Quota policies of type calendar. Indicates the ISO 8601 date and time when the Quota counter begins counting (regardless of whether or not any requests have been received from any apps.)

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

XML File Name	Description
	Valid value: ISO 8601 formatted date and time, for example `2013-02-0500:00:00`.
Interval (Required)	Specifies the interval of time (in hours, minutes, or days as defined by TimeUnit) applicable to the Quota. For example, a value of 24 for the Interval with aTimeUnit of hours means that the Quota is calculated over one day (24 hours). Valid value: integer
TimeUnit (Required)	Valid values: second, minute, hour, day, or month **Note** Distributed Quota does not support seconds for the TimeUnit.
Allow count (Required)	Specifies a message count for the quota. For example, a value of 100 for the Allow count with duration of 1 month means that the quota is set to be 100 messages per month. Valid value: integer **Note** If a count reference is specified, it takes precedence over the Allow count value. **Example** The element <Allow count="2000" countRef="request.header.allowed_quota"/> has a count header (countRef="request.header.allowed_quota") along with the count value of 2000.
Identifier (Optional)	Variable used for uniquely identifying the client app.
MessageWeight (Optional)	Specifies the weighting defined for each message. Message weight is used to increase impact of request messages that, for example, consume more computational resources than others. For example, you may want calculate POST messages as being twice as expensive as GET messages. A value representing `MessageWeight` can be extracted from HTTP headers, query parameters, or an XML or JSON request payload. For example, if the Quota is 10 messages per minute and the MessageWeight for POST requests is 2, then only 5 POST requests are permitted in any 10 minute interval. The Quota is violated (that is, messages rejected) starting with the sixth message.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

XML File Name	Description
Distributed	When set to `true`, a central counter is maintained that is continuously updated by all SAP API Management servers. Example If the limit is 10 messages/hour and number of servers assigned is 2, then the Quota count total is maintained for both servers. Valid values: true or false
PreciseAtSecondsLevel	The default precision for Quotas intervals is one minute. When set to true, Quota precision is set to record at intervals of one second. Use this setting when you have a Quota that uses minutes as the TimeUnit, and you need to ensure that Quotas are counted and enforced by seconds. Valid values: true or false
Synchronous	This setting determines how the distributed Quota counter is updated. If set to true, the quota counter is updated in the central repository synchronously. This means that the update is made at the same time the API call is quota-checked. When synchronous is set to true, you are guaranteed that no API calls over the quota will be allowed. If set to false, the quota counter is updated asynchronously, and it is possible that some API calls will go through over the quota, depending on when the repository was asynchronously updated. The default asynchronous update interval is 10 seconds. Use AsynchronousConfiguration field, described below, to configure the asynchronous behavior. The primary reason to choose asynchronous updating (false) is that it is slightly less efficient to enforce synchronous updates. If you can accept the potential performance impact, or expect lower throughput, so that performance will not be an issue, you may wish to set synchronous to be true. If it is essential to avoid going over the quota, then set this field to true. Valid values: true or false (default)

AsynchronousConfiguration (Optional)	This configuration element is only required when Synchronous is set to false. This element enables you to configure the synchronization interval among distributed Quota counters. The default asynchronous update interval is 10 seconds. You can modify this by adding the child element SyncIntervalInSeconds.
	 Example
	`<Synchronous>false</Synchronous>`
	`<AsynchronousConfiguration>`
	
	`<SyncIntervalInSeconds>20</SyncIntervalInSeconds>`

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

XML File Name	Description

	`</AsynchronousConfiguration>`
	
	Alternatively, you can use the SyncMessageCount option instead, but you cannot use both. `SyncMessageCount` specifies the number of requests across all SAP API Management message processors between quota updates. The following example specifies that the quota count is updated every 5 requests across all API Management message processors:
	`<Synchronous>false</Synchronous>`
	`<AsynchronousConfiguration>`
	`<SyncMessageCount>5</SyncMessageCount>`
	`</AsynchronousConfiguration>`

8.2.2.6 Policy-specific Variables

The following predefined Flow variables are automatically populated when a Quota policy executes. For more information about Flow variables, see Variables reference.

Variables	Type	Permissions	Description
rate limit.{policy_name}.allowed.count	Long	Read-only	Returns the allowed quota count
ratelimit.{policy_name}.used.count	Long	Read-only	Returns the current quota used within a quota interval
ratelimit.{policy_name}.available.count	Long	Read-only	Returns the available quota count in the counter
ratelimit.{policy_name}.exceed.count	Long	Read-only	Returns the count that exceeds the limit in the current counter
ratelimit.{policy_name}.total.exceed.count	Long	Read-only	Returns the total count that exceeds the limit, because the quota policy is attached
ratelimit.{policy_name}.expiry.time	Long	Read-only	Returns the time in milliseconds, which determines when the quota expires and new quota counter starts

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Variables	Type	Permissions	Description
ratelimit.{policy_name}.identifier	String	Read-only	Returns the (client) identifier reference attached to the policy
ratelimit.{policy_name}.class	String	Read-only	Returns the class associated with the client identifier
ratelimit.{policy_name}.class.allowed.count	Long	Read-only	Returns the allowed quota count defined in the class
ratelimit.{policy_name}.class.used.count	Long	Read-only	Returns the used quota within a class
ratelimit.{policy_name}.class.available.count	Long	Read-only	Returns the available quota count in the class
ratelimit.{policy_name}.class.exceed.count	Long	Read-only	Returns the count that exceeds the limit in the class
ratelimit.{policy_name}.class.total.exceed.count	Long	Read-only	Returns the total count that exceeds the limit in the class, because the Quota policy is attached

8.2.2.7 Policy-specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

Error Code	Message
QuotaViolation	Rate limit quota violation.Quota limit: {0} exceeded by {1}. Total violation count :{ 2}. {3}
InvalidMessageWeight	Invalid message weight value{0}
ErrorLoadingProperties	Error loading rate limit properties from {0}
FailedToResolveQuotaIntervalReference	Failed to resolve quota interval reference {0} inquota policy {1}

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Error Code	Message
InvalidQuotaInterval	Invalid quota interval {0} inquota policy {1}
FailedToResolveQuotaIntervalTimeUnitReference	Failed to resolve quota time unit reference {0} in quota policy {1}
InvalidQuotaTimeUnit	Invalid quota time unit {0}in quota policy {1}
InvalidTimeUnitForDistributedQuota	Invalid timeunit {0} fordistributed quota
InvalidSynchronizeIntervalForAsyncConfiguration	'SyncIntervalInSeconds'should be a value greater than zero
InvalidSynchronizeMessageCountForAsyncConfiguration	'SyncMessageCount' should be a value greater than zero
InvalidAsynchronizeConfigurationForSynchronousQuota	AsynchronousConfiguration is not valid for synchronous quota
InvalidQuotaType	Invalid Quotatype:{0};ValidTypes are calendar,rollingwindow,flexi
InvalidStartTime	Invalid Starttime:{0}; StartTime should be of the format'yyyy-MM-dd HH:mm:ss
StartTimeNotSupported	Start time is not supported for quotatype {0}. Starttime is supported only for'calendar' based type

8.2.3 Reduce latency using ResponseCache

- Attaching a ResponseCache policy
- Configuring the ResponseCache policy
- Configuring SAP API Management to use HTTP cache headers
- ResponseCache Flow variables
- Policy-specific error codes

ResponseCache can be configured to save and periodically refresh copies of response messages. As apps make requests to the same URI, SAP API Management can be configured to return cached responses, rather than forwarding those requests to the backend server.

ResponseCache is commonly used when backend data is updated only periodically. For example, an API that exposes weather report data might only refresh the weather data once every ten minutes. By returning cached responses and refreshing once every ten minutes, ResponseCache decreases the number of request reaching the backend, and also eliminates a network hop for a significant portion of requests to the API.

There are two main benefits to ResponseCache:

- Reduced latency: The request is satisfied from the cache (removing a network hop), so the response is returned in a shorter time, making the API more responsive.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- Reduced network traffic: Representations are reused, so the impacts of processing duplicate or redundant requests are reduced. ResponseCache reduces the amount of bandwidth consumed overall.

Unlike other policies, a ResponseCache policy must be attached to both request and response paths within a Flow.

> **Note**
>
> You can configure the ResponseCache proxy to also look at certain HTTP response caching headers and take appropriate actions according to the directives of those headers. For example, on responses from backend targets, SAP API Management supports the Cache-Control header, which can be used to control the maximum age of a cached response, among other directives. For more information, see HTTP response caching.

Cached response messages are returned to requesting apps based on request message keys configured in the policy. At runtime, SAP API Management inspects request messages for a 'key' that you specify. A key is a piece of information (such as query parameter) common to all request messages whose responses you need to cache. When a key match occurs, SAP API Management returns the cached response, instead of forwarding the request to the backend service.

8.2.3.1 Samples

10-minute cache

This sample shows how to have cached responses kept for 10 minutes.

Imagine that you have an API at the following URL:

```
http://{org-name}-{env}.<host:port>/weather/forecastrss?w=23424778
```

You're using the query parameter w as a cache key. SAP API Management checks the value of the query parameter w whenever a request is received. If a valid (that is, non-expired) response is present in the cache, then the cached response message is returned to the requesting client.

Now imagine that you have a ResponseCache policy configured as follows:

```
<ResponseCache name="ResponseCache">
    <CacheKey>
        <KeyFragment ref="request.queryparam.w" />
    </CacheKey>
    <ExpirySettings>
        <TimeoutInSec>600</TimeoutInSec>
    </ExpirySettings>
</ResponseCache>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The first time the API proxy receives a request message for the following URL, the response is cached. On the second request within 10 minutes, a cache lookup occurs -- the cached response is returned to the app with no request forwarded to the backend service.

`http://{org-name}-{env}.<host:port>/weather/forecastrss?w=23424778`

.

> **Note**
>
> HTTP headers and query parameters are automatically populated as variables when a request is received. Any HTTP header is available as `request.header.{header_name}`, for example `request.header.user-agent.` Any query parameter is available as `request.queryparam.{queryparam_name}`, for example `request.queryparam.action`.

8.3 Element reference

The element reference describes the elements and attributes of the Response Cache policy.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResponseCache async="false" continueOnError="false" enabled="true" name="Response-Cache-1">
    <DisplayName>Response Cache 1</DisplayName>
    <FaultRules/>
    <Properties/>
    <CacheKey>
        <Prefix/>
        <KeyFragment ref="request.uri" />
    </CacheKey>
    <Scope>Exclusive</Scope>
    <ExpirySettings>
        <ExpiryDate/>
        <TimeOfDay/>
        <TimeoutInSec ref="flow.variable.here">300</TimeoutInSec>
    </ExpirySettings>
    <CacheResource>cache_to_use</CacheResource>
    <CacheLookupTimeoutInSeconds/>
    <ExcludeErrorResponse/>
    <SkipCacheLookup/>
    <SkipCachePopulation/>
    <UseAcceptHeader/>
    <UseResponseCacheHeaders/>
</ResponseCache>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.4 <ResponseCache> attributes

```
<ResponseCache async="false" continueOnError="false" enabled="true" name="Response-Cache-1">
```

The following attributes are common to all policy parent elements.

Field Name	Description	Default	Required
name	The unique name of the policy. Use this name to reference the policy within a Step definition element. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the `<DisplayName>` element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Yes
continueOnError	Set to `false` to return an error when a policy fails. This is expected behavior for most policies. Set to `true` to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to `true` to enforce the policy. Set to `false` to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to `true`, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to `true` improves API proxy performance. However, overusing async can hurt performance with too much thread switching. To use asynchronous behavior in API proxies, see Java Callout Policy.	false	Optional

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.4.1.1.1 <DisplayName> Element

You can use in addition to the name attribute to label the policy in the SAP API Management UI proxy editor with a different, natural-language name.

```
<DisplayName>Policy Display Name</DisplayName>
```

Default:	N/A If you omit this element, the the value of the policy's name attribute is used.
Presence:	Optional
Type:	String

8.4.1.1.2 <CacheKey> Element

Configures a unique pointer to a piece of data stored in the cache.

> **Note**
>
> You can further augment the cache key configured here by using values from response headers. For more information, see ResponseCache Flow Variables.

```
<CacheKey>
    <Prefix>string</Prefix>
    <KeyFragment ref="variable_name" />
    <KeyFragment>literal_string</KeyFragment>
</CacheKey>
```

<CacheKey> constructs the name of each piece of data stored in the cache.

CacheKey constructs the name of each piece of data stored in the cache. The cache key you define in the `PopulateCache` policy is the name that you use in the `LookupCache` policy to get the specific entry that you need from the cache.

At runtime, <KeyFragment> values are prepended with either the Scope element value or Prefix value. For example, the following results in a cache key of `UserToken__apiAccessToken__<value_of_client_id>`:

```
<CacheKey>
    <Prefix>UserToken</Prefix>
    <KeyFragment>apiAccessToken</KeyFragment>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <KeyFragment ref="request.queryparam.client_id" />
</CacheKey>
```

You use the <CacheKey> element in conjunction with <Prefix> and <Scope>. For more information, see Working with cache keys.

8.4.1.1.3 <CacheLookupTimeoutInSeconds> element

This element specifies the number of seconds after which an unsuccessful cache lookup will be considered a cache miss. If this occurs, flow resumes along the cache-miss path.

```
<CacheLookupTimeoutInSeconds>30</CacheLookupTimeoutInSeconds>
```

Default:	30
Presence:	Optional
Type:	Integer

8.4.1.1.4 <CacheResource> Element

This element specifies the cache where messages should be stored.

Omit this element completely if this policy (and your corresponding LookupCache and InvalidateCache policies) is using the included shared cache. For more information, see Manage caches for an environment.

```
<CacheResource>cache_to_use</CacheResource>
```

Default:	N/A
Presence:	Optional
Type:	String

For more about configuring caches, see Creating and Editing an Environment Cache.

8.4.1.1.5 <ExcludeErrorResponse> element

Currently, by default, this policy caches HTTP responses with any possible Status code. That means both success and error responses are cached. For example, responses with both 2xx and 3xx status codes are cached by default.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Set this element to true if you do not wish to cache target responses with HTTP error status codes; only responses with status codes from 200 to 205 will be cached if this element is true. These are the only HTTP status codes that SAPI API Management counts as "success" codes, and you cannot change this association.

> **Note**
>
> In a future release (to be determined), the default setting of this element will change to true.

```
<ExcludeErrorResponse>true</ExcludeErrorResponse>
```

Default:	false
Presence:	Optional
Type:	Boolean

8.4.1.1.6 <ExpirySettings> / <ExpiryDate> element

Specifies the date on which a cache entry should expire. Use the form mm-dd-yyyy. When present, this element's sibling, `<TimeoutInSec>`, overrides `<ExpiryDate>`.

```
<ExpirySettings>
    <ExpiryDate ref="{date_variable}">expiration_date</ExpiryDate>
</ExpirySettings>
```

Default:	N/A
Presence:	Optional
Type:	String

Attributes

```
<ExpiryDate ref="" />
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Attribute	Description	Default	Presence	Type
ref	The variable from which to get the value. Should not be used if this element contains a literal value.	N/A	Optional	String

8.4.1.1.7 <ExpirySettings> element

This element specifies when a cache entry should expire. When present, `<TimeoutInSec>` overrides both `<TimeOfDay>` and `<ExpiryDate>`.

```
<ExpirySettings>
  <TimeOfDay ref="time_variable">expiration_time</TimeOfDay>
  <TimeoutInSec ref="duration_variable">seconds_until_expiration</TimeoutInSec>
  <ExpiryDate ref="date_variable">expiration_date</ExpiryDate>
</ExpirySettings>
```

Default: N/A

Presence: Required

Type: N/A

8.4.1.1.8 <CacheKey> / <KeyFragment> Element

This element specifies a value that should be included in the cache key, creating a namespace for matching requests to cached responses.

```
<KeyFragment ref="variable_name"/>
<KeyFragment>literal_string</KeyFragment>
```

Default: N/A

Presence: Optional

Type: N/A

This can be a key (a static name that you provide) or a value (a dynamic entry set by referencing a variable). All specified fragments combined (plus the prefix) are concatenated to create the cache key.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<KeyFragment>apiAccessToken</KeyFragment>
<KeyFragment ref="request.queryparam.client_id" />
```

You use the <KeyFragment> element in conjunction with <Prefix> and <Scope>. For more information, see Working with Cache Keys

Attributes

Attribute	Type	Default	Required	Description
ref	string		No	The variable from which to get the value. Should not be used if this element contains a literal value.

8.4.1.1.9 <CacheKey> / <Prefix> Element

This element specifies a value to use as a cache key prefix.

```
<Prefix>prefix_string</Prefix>
```

Default: N/A

Presence: Optional

Type: String

You can use this value instead of <Scope> when you want to specify your own value rather than a <Scope> - enumerated value. If defined, <Prefix> prepends the cache key value for entries written to the cache. A <Prefix> element value overrides a <Scope> element value.

You use the <Prefix> element in conjunction with <CacheKey> and <Scope>. For more information, see Working with cache keys.

8.4.1.1.10 <Scope> Element

Enumeration used to construct a prefix for a cache key when a <Prefix> element is not provided in the <CacheKey> element.

```
<Scope>scope_enumeration</Scope>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The <Scope> setting determines a cache key that is prepended according to the <Scope> value. For example, a cache key would take the following form when the scope is set to Exclusive:

`orgName__envName__apiProxyName__deployedRevisionNumber__proxy|TargetName__ [ serialize dCacheKey ].`

If a **<Prefix>** element is present in **<CacheKey>**, it supercedes a **<Scope>** element value. Valid values include the enumerations below.

You use the **<Scope>** element in conjunction with **<CacheKey>** and **<Prefix>.** For more information, see Working with cache keys.

Scope values

Global	Cache key is shared across all API proxies deployed in the environment. Cache key is prepended in the form `orgName __ envName __.` If you define a **<CacheKey>** entry with the **<KeyFragment>** apiAccessToken and a **<Global>** scope, each entry is stored as orgName__envName__apiAccessToken, followed by the serialized value of the access token. For an API proxy deployed in an environment called 'test' in an organization called 'apifactory', access tokens would be stored under the following cache key: apifactory__test__apiAccessToken.
Application	API proxy name is used as the prefix. Cache key is prepended in the form `orgName__envName__apiProxyName`.
Proxy	ProxyEndpoint configuration is used as the prefix. Cache key is prepended in the form `orgName__envName__apiProxyName__deployedRevisionNumber__proxyEndpointName`.
Target	TargetEndpoint configuration is used as the prefix. Cache key prepended in the form `orgName__envName__apiProxyName__deployedRevisionNumber__targetEndpointName`.
Exclusive	Default. This is the most specific, and therefore presents minimal risk of namespace collisions within a given cache. Prefix is one of two forms: · If the policy is attached to the `ProxyEndpoint` flow, prefix is of the form ApiProxyName_ProxyEndpointName. · If the policy is attached at `TargetEndpoint`, prefix is of the form ApiProxyName_TargetName. Cache key prepended in the form · `orgName__envName__apiProxyName__deployedRevisionNumber__proxyNameITargetName` For example, the full string might look like this: apifactory__test__weatherapi__16__default__apiAccessToken .

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.4.1.1.11 <SkipCacheLookup> element

Defines an expression that, if it evaluates to true at runtime, specifies that cache lookup should be skipped and the cache should be refreshed.

```
<SkipCacheLookup>variable_condition_expression</SkipCacheLookup>
```

Default:	N/A
Presence:	Optional
Type:	String

From the following example, if the bypass-cache variable is set to true in an incoming header, cache lookup is skipped and the cache is refreshed.

```
<SkipCacheLookup>request.header.bypass-cache = "true"</SkipCacheLookup>
```

8.4.1.1.12 <SkipCachePopulation> element

Defines an expression that, if it evaluates to true at runtime, specifies that a write to the cache should be skipped.

```
<SkipCachePopulation>variable_condition_expression</SkipCachePopulation>
```

Default:	N/A
Presence:	Optional
Type:	String

For example, the following would skip the cache write if the response status code was 400 or higher:

```
<SkipCachePopulation>response.status.code >= 400</SkipCachePopulation>
```

8.4.1.1.13 <ExpirySettings> / <TimeOfDay> element

The time of day at which a cache entry should expire. Use the form hh:mm:ss . When present, this element's sibling, `<TimeoutInSec>`, overrides `<TimeOfDay>`.

Enter time of day in the format HH:mm:ss, where HH represents the hour on a 24-hour clock. For example, 14:30:00 for 2:30 in the afternoon.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

For the time of day, the default locale and time zone will vary depending on where the code is running (which isn't knowable when you configure the policy). For information on configuring your locale, see Manage caches for an environment.

```
<ExpirySettings>
    <TimeOfDay ref="time_variable">expiration_time</TimeOfDay>
</ExpirySettings>
```

Default:

Presence: Optional

Type: String

Attributes

Attribute	Description	Default	Presence	Type
ref	Variable with the expiration time value.	N/A	Optional	String

8.4.1.1.14 <ExpirySettings> / <TimeoutInSec> element

The number of seconds after which a cache entry should expire. When present, this element overrides its siblings, `<TimeOfDay>` and `<ExpiryDate>`.

```
<ExpirySettings>
    <TimeoutInSec ref="duration_variable">seconds_until_expiration</TimeoutInSec>
</ExpirySettings>
```

Default: N/A

Presence: Optional

Type: Integer

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Attributes

Attribute	Description	Default	Presence	Type
ref	Variable with the timeout value.	N/A	Optional	String

8.4.1.1.15 <UseAcceptHeader> element

Set to true to have a reponse cache entry's cache key appended with values from response Accept headers.

SAP API Management uses the Accept, Accept-Encoding, Accept-Language and Accept-Charset request headers when calculating the cache key. This approach prevents a client from getting a media type they did not ask for.

For example, consider if two requests come in from the same URL, where the first request accepts gzip and the second does not. The first request will be cached, and the cached entry will (probably) be a gzipped response. The second request will read the cached value and may then return a gzipped entry to a client that is not capable of reading gzip.

See ResponseCache Flow Variables for more.

```
<UseAcceptHeader>false</UseAcceptHeader>
```

Default: false

Presence: Optional

Type: Boolean

8.4.1.1.16 <UseResponseCacheHeaders> element

Set to `true` to have HTTP response headers considered when setting the "time to live" (TTL) of the response in the cache. When this is `true`, SAP API Management considers the values of the following response headers, comparing the values with those set by `<ExpirySettings>` when setting time to live:

- `Cache-Control s-maxage`
- `Cache-Control max-age`
- `Expires`

```
<UseResponseCacheHeaders>false</UseResponseCacheHeaders>
```

Default: false

Presence: Optional

Type: Boolean

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.4.1.2 Configuring SAP API Management to Use HTTP Cache Headers

The following policy configuration uses UseAcceptHeader and UseResponseCacheHeaders elements to instruct SAP API Management to use Accept-* headers calculate cache keys and to consider HTTP response cache headers like Cache-Control. For more information on how SAP API Management uses HTTP cache headers, see HTTP response caching

```
<ResponseCache name="response_cache">
    <UseAcceptHeader>true</UseAcceptHeader>
    <UseResponseCacheHeaders>true</UseResponseCacheHeaders>
    <CacheResource>cacheresource</CacheResource>
    <CacheKey>
        <KeyFragment ref="request.header.key1"/>
    </CacheKey>
</ResponseCache>
```

8.4.1.3 ResponseCache Flow Variables

The following predefined Flow variables are populated when a ResponseCache policy is executed. For more information about Flow variables, see Variables reference.

Variables	Type	Permission	Description
responsecache.<policy-name>.cachename	String	Read-only	Returns the cache used in the policy
responsecache.<policy-name>.cachekey	String	Read-only	Returns the key used
responsecache.<policy-name>.cachehit	Boolean	Read-only	True if the policy execution is successful
responsecache.<policy-name>.invalidentry	Boolean	Read-only	True if the cache entry in not valid

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.4.1.4 Policy-specific Error Codes

Error Code	Message
CacheNotFound Cache	{0}, not found in organization : {1}
InvalidCacheResourceReference	Invalid cache resource reference {0} inStep definition {1}. Context {2}
EntryCannotBeCached Entry	{0} can not be cached. Onlyserializable entries are cached.
CannotDeleteStepDefinition	Step definition {0} should be detached before deletion
ResponseCacheStepAttachmentNotAllowedReq	Response cache step definition {0} cannot be attached more than once in the request path
ResponseCacheStepAttachmentNotAllowedResp	Response cache step definition {0} cannot be attached more than once in the response path
InvalidTimeout	CacheLookupTimeoutInSeconds{0} value should be greater than zero

8.4.2 Shield APIs Using SpikeArrest

- Configuring a SpikeArrest policy
- Element reference
- SpikeArrest Flow variables
- Policy-specific error codes

A significant challenge to the maintenance of healthy APIs is traffic control. APIs consumed by client apps are vulnerable to performance lags and downtime caused by the sudden influx of request messages, whether caused by malicious attacks, buggy apps, or seasonality.

Policies of type SpikeArrest throttle the number of requests forwarded from the point in the processing Flow where the Policy is attached as a processing Step. You can attach a SpikeArrest policy at the ProxyEndpoint request Flow to limit inbound requests. You can also attach the SpikeArrest policy at the TargetEndpoint request Flow to limit request forwarded to the backend service.

Unlike Quotas, spike arrests are not implemented as counts. Rather, they are implemented as a rate limit which is based on the time the last matching message was processed.

If you specify 5 messages per minute, it means that requests can only be submitted at the rate of one per 12 sec (1/5 minute) interval. A second request within 12 seconds on the same SAP API Management server will be rejected. Even with a larger number (100 per second), if two requests come in nearly simultaneously to the same SAP API Management server, one will be rejected. Each successful (non-arrested) request will update the spike arrest's last processed count.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

No counter is maintained for spike arrests, only a time that the last message was successfully passed through the SpikeArrest policy. On a given SAP API Management server, if a request is received now, all subsequent requests will fail until 12 seconds has elapsed.

Because Spike Arrest is not distributed, you might see some discrepancy between the actual behavior of the system and your expected results. In general, you should use SpikeArrest to set a limit that throttles traffic to what your backend services can handle. Do not use SpikeArrest to limit traffic from individual clients.

SAP API Management supports both SpikeArrest and Quota. The two policy types support different, though related, use cases:

- SpikeArrest smooths inflow request patterns over short intervals to ensure that demand does not outstrip capacity.
- Quota limits the number of API requests that each app can submit over a longer time interval. Quotas are generally tied to an app's consumer key, ensuring that a specific app is limited to an approved number of requests.
- SpikeArrest is not distributed. Quota counters are distributed. However, Quota counters support, at a minimum, an interval of one minute.

It is very common to combine SpikeArrest and Quota policies--the SpikeArrest prevents bursts over short intervals, while Quotas enforce longer term consumption limits.

Always make sure that you do performance testing in your test environment to ensure that proper behavior from your traffic management policies before you release to prod.

For more information on the Quota policy type, see Rate limit API traffic using Quota.

> **Note**
>
> The name attribute for this policy is restricted to these characters: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.

8.4.2.1 Samples

The following samples illustrate how you can use the SpikeArrest policy as a protection against traffic spikes.

Per second

It refers to 5 per second. This policy smoothes the rate to 1 request allowed every 200 milliseconds (1000 / 5).

```
<SpikeArrest name="SpikeArrest">
  <Rate>5ps</Rate>
</SpikeArrest>
```

Per minute

It refers to 12 per minute. The policy smoothes the rate to 1 request allowed every 5 seconds (60 / 12).

```
<SpikeArrest name="SpikeArrest">
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  <Rate>12pm</Rate>
</SpikeArrest>
```

With message weight

It refers to 12 per minute (1 request allowed every 5 seconds, 60 / 12), with message weight that provides additional throttling on specific clients or apps (captured by the Identifier).

```
<SpikeArrest name="SpikeArrest">
  <Rate>12pm</Rate>
  <Identifier ref="client_id" />
  <MessageWeight ref="request.header.weight" />
</SpikeArrest>
```

Rate from variable

Setting rate with a `variable` in the request. The variable value must be in the form of {int}pm or {int}ps.

```
<SpikeArrest name="SpikeArrest">
  <Rate ref="request.header.rate" />
</SpikeArrest>
```

8.4.2.2 Element reference

The element reference describes the elements and attributes of the SpikeArrest policy.

```
<SpikeArrest async="false" continueOnError="false" enabled="true" name="Spike-Arrest-
1">
    <DisplayName>Custom label used in UI</DisplayName>
    <Rate>30ps</Rate>
    <Identifier ref="request.header.some-header-name"/>
    <MessageWeight ref="request.header.weight"/>
</SpikeArrest>
```

8.4.2.2.1 <SpikeArrest> attributes

```
<SpikeArrest async="false" continueOnError="false" enabled="true" name="Spike-Arrest-
1">
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.4.2.2.2 Configuring a SpikeArrest Policy

Configure your SpikeArrest policy using the following elements.

Field Name	Description	Default	Required
name	The unique name of the policy. Use this name to reference the policy within a Step definition element. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the `<DisplayName>` element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Yes
continueOnError	Set to `false` to return an error when a policy fails. This is expected behavior for most policies. Set to `true` to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to `true` to enforce the policy. Set to `false` to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to `true`, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to `true` improves API proxy performance. However, overusing async can hurt performance with too much thread switching. To use asynchronous behavior in API proxies, see Java Callout Policy.	false	Optional

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.4.2.2.3 <DisplayName> element

You can use this element in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

```
<DisplayName>Policy Display Name</DisplayName>
```

Default:	N/A If you omit this element, the the value of the policy's name attribute is used.
Presence:	Optional
Type:	String

8.4.2.2.4 <Rate> element

This element pecifies the rate at which to limit traffic spikes (or bursts). Specify a number of requests that are allowed in per minute or per second intervals. However, keep reading for a description of how the policy behaves at runtime to smoothly throttle traffic.

```
<Rate>10ps</Rate>
<Rate>30pm</Rate>
<Rate ref="request.header.rate" />
```

Default	N/A
Presence	Required
Type	Integer
Valid values	{int}ps (number of requests per second, smoothed into intervals of milliseconds) {int}pm (number of requests per minute, smoothed into intervals of seconds) Note: In rate smoothing, the number of requests is always a whole number greater than zero. Smoothing never involves calculating fractions of requests.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Attributes

Attribute	Description	Default	Presence
ref	A reference to the variable containing the rate setting, in the form of {int}pm or {int}ps.	N/A	Optional

8.4.2.2.5 <Identifier> element

You can Use the `<Identifier>` element to uniquely identify and apply spike arrest against individual apps or developers. You can use a variety of variables to indicate a unique developer or app, whether you're using custom variables or predefined variables, such as those available with the Verify API Keys Using API Key Validation. See also the Variables Reference.

Use in conjunction with `<MessageWeight>` for more fine-grained control over request throttling.

If you don't use this element, all calls made to the API proxy are counted for spike arrest.

```
<Identifier ref="client_id"/>
```

Default	N/A
Presence	Optional
Type	String

Attributes

Attribute	Description	Default	Presence
ref	A reference to the variable containing the data that identifies the app or developer.	N/A	Required

8.4.2.2.6 <MessageWeight> element

You can use in conjunction with `<Identifier>` to further throttle requests by specific clients or apps.

This element specifies the weighting defined for each message. Message weight is used to modify the impact of a single request on the calculation of the Spike Arrest limit. Message weight can be set by variables based on HTTP

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

headers, query parameters, or message body content. For example, if the Spike Arrest Rate is 10pm, and an app submits requests with weight 2, then only 5 messages per minute are permitted from that app.

```
<MessageWeight ref="request.header.weight"/>
```

Default	N/A
Presence	Optional
Type	Integer

Attributes

Attribute	Description	Default	Presence
ref	A reference to the variable containing the message weight for the specific app or client.	N/A	Required

Note

- In general, you should use SpikeArrest to set a limit that throttles traffic to what your backend services can handle.
- No counter is maintained for SpikeArrests, only a time that the last message was successfully passed through the SpikeArrest policy.

8.4.2.3 SpikeArrest Flow Variables

When a Spike Arrest policy executes, the following Flow variables are populated.

For more information about Flow variables, see Variables Reference.

Variable	Type	Permission	Description
ratelimit.{policy_name}.allowed.count	Long	Read-only	Returns the allowed limit count.
ratelimit.{policy_name}.used.count	Long	Read-only	Returns the limit used in the counter.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Variable	Type	Permission	Description
ratelimit.{policy_name}.exceed.count	Long	Read-only	Returns the count exceeds the limit in the current counter.
ratelimit.{policy_name}.expiry.time	Long	Read-only	Returns the time in milliseconds based on which the limit expires and new counter starts.

8.4.2.4 Policy-specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

Error Code	Message
SpikeArrestViolation	Spike arrest violation. Allowed rate : {0}
InvalidMessageWeight	Invalid message weight value {0}
ErrorLoadingProperties	Error loading rate limit properties from {0}
InvalidAllowedRate	Invalid spike arrest rate {0}.
FailedToResolveSpikeArrestRate	Failed to resolve Spike Arrest Rate reference {0} in SpikeArrest policy {1}

SAP API Management organizations can be configured to return an HTTP status code of 429 (Too Many Requests) for all requests that exceed a rate limit set by a SpikeArrest policy. The default configuration returns an HTTP status code of 500 (Internal Server Error).

Contact SAP API Management to have the `features.isHTTPStatusTooManyRequestEnabled` property set to `true` for organizations for which you want Spike Arrest policy violations to return an HTTP status code of 429.

SAP API Management for Private Cloud customers can set this property with the following API call:

```
curl -u email:password -X POST -H "Content-type:application/xml"
http://host:8080/v1/o/myorg -d \
"<Organization type="trial" name="MyOrganization">
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <DisplayName>MyOrganization</DisplayName>
    <Environments/>
    <Properties>
        <Property name="features.isHTTPStatusTooManyRequestEnabled">true</Property>
    </Properties>
</Organization>"
```

8.4.3 Reset Quota Counter Using ResetQuota

- Configuring a ResetQuota policy
- Policy-specific error codes

The ResetQuota policy type enables you to dynamically reset the limit for a specified Quota.

Example

You can use this policy to reset a developer's Quota counter when additional API calls are purchased.

Note

The name attribute for this policy is restricted to these characters: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.

8.4.3.1 Configuring a ResetQuota policy

Configure your ResetQuota policy using the following elements.

Field Name	Description
Quota Name	Specifies the name of the Quota policy whose counter should be reset.
Identifier name	Variable used for uniquely identifying the client, for example client_id.
Allow (Allow integer)	Specifies the message count to which the Quota will be reset.

Example

Reset Quota Examples

Reset top-level Quota.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<ResetQuota name="resetQuota">
 <Quota name="request.header.quotapolicy">
  <Identifier name="_default">
    <Allow>100</Allow>
  </Identifier>
 </Quota>
</ResetQuota>

<ResetQuota name="resetQuota">
 <Quota name="request.header.quotapolicy">
  <Identifier name="_default">
    <Allow ref="request.header.allowquota" />
  </Identifier>
 </Quota>
</ResetQuota>
```

Reset top-level Quota for different identifiers.

```
<ResetQuota name="resetQuota">
 <Quota ref="request.header.quotapolicy">
  <Identifier ref="request.header.identifier">
    <Allow>100</Allow>
  </Identifier>
 </Quota>
</ResetQuota>
```

Reset class-level Quotas.

```
<ResetQuota name="resetQuota">
 <Quota name="quotapolicy">
  <Identifier name="_default">
   <Class ref="request.header.classIdentifier">
    <Allow>200</Allow>
  </Identifier>
  </Identifier>
 </Quota>
</ResetQuota>
```

8.4.3.2 Policy-specific Error Codes

The default format for error codes returned by Policies is:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

Error Code	Message
InvalidRLPolicyDefinition	Invalid rate limit policy {0}
NoRLPolicy	Quota policy {0} is not attached.
InvalidCount	Invalid count value {0} for identifier {1} in {2}
FailedToResolveAllowCountRef	Failed to resolve allow count reference {0} foridentifier {1} in {2}

8.4.4 Throttle Backend Connections Using ConcurrentRatelimit

The ConcurrentRatelimit policy enables you to throttle inbound connections to your backend services from API proxies running on SAP API Management. In distributed environments, app traffic may be managed by many replicated API proxies. While each API proxy might be handling just a few connections, collectively, a set of replicated API proxies, all of which point to the same backend service, might swamp the capacity of the service to which they forward requests.

When the connection limit is exceeded, additional requests return HTTP response code 503:

```
503 Service Unavailable
```

8.4.4.1 Policy Attachment

The ConcurrentRatelimit policy must be attached as a Step to three Flows on a TargetEndpoint: request, response, and DefaultFaultRule. (A validation error will be thrown at deployment time if the policy is attached to any other Flows, including any ProxyEndpoint Flows.)

Note that when an API proxy is re-deployed, the counter values are reset.

Example

To attach a ConcurrentRatelimit policy called ConnectionThrottler to a TargetEndpoint called MyTargetEndpoint, create the following TargetEndpoint configuration:

```
<TargetEndpoint name="MyTargetEndpoint">
  <DefaultFaultRule name="DefaultFaultRule">
    <Step><Name>ConnectionThrottler</Name></Step>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <AlwaysEnforce>true</AlwaysEnforce>
  </DefaultFaultRule>
  <PostFlow name="PostFlow">
    <Response>
      <Step><Name>ConnectionThrottler</Name></Step>
    </Response>
  </PostFlow>
  <PreFlow name="PreFlow">
    <Request>
      <Step><Name>ConnectionThrottler</Name></Step>
    </Request>
  </PreFlow>
  <HTTPTargetConnection>
    <URL>http://api.mybackend.service.com</URL>
  </HTTPTargetConnection>
</TargetEndpoint>
```

> **Note**
>
> Counters are reset when the API proxy is redployed.

8.4.4.2 Configuring a ConcurrentRatelimit Policy

Configure the ConcurrentRatelimit policy using the following elements.

Field Name	Description
name	The unique name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %.
AllowConnections	The number of concurrent connections between SAP API Management and a backend service that are allowed at any given time. The optional attribute ttl can be added to this element to cause the counter to automatically decrement after the number of seconds configured. This can clean up any connections that were not decremented properly in the response path.
isDistributed	A boolean that determines whether counter values are shared across instances of SAP API Management's server infrastructure.
TargetIdentifier	The name of the TargetEndpoint to which the throttling should be applied.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.4.4.3 Policy-specific Flow Variables

A set of pre-defined Flow variables are populated each time the policy executes:

- `concurent.ratelimit.{policy_name}.allowed.count`
- `concurent.ratelimit.{policy_name}.used.count`
- `concurent.ratelimit.{policy_name}.available.count`
- `concurent.ratelimit.{policy_name}.identifier`

8.4.4.4 Policy-specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

Error Code	Message
ConcurrentRatelimtViolation	ConcurrentRatelimit connection exceeded.Connection limit : {0}
InvalidCountValue	ConcurrentRatelimit invalid count value specified.
ConcurrentRatelimitStepAttachmentNotAllowedAtProxyEndpoint	Concurrent Ratelimit policy {0} attachment is not allowed at proxy request / response / fault paths
ConcurrentRatelimitStepAttachmentMissingAtTargetEndpoint	Concurrent Ratelimitpolicy {0} attachment is missing in target request / response / fault paths
InvalidTTLForMessageTimeOut	ConcurrentRatelimit invalid ttl value specified for message timeout

8.5 Security Policies

8.5.1 Authorize Requests Using OAuth 1.0a

OAuth 1.0a defines a standard protocol that enables app users to authorize apps to consume APIs on their behalf, without requiring app users to disclose their passwords to the app in the process. SAP API Management enables

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

you to protect APIs in a way that ensures that an app uses the authorized app to consume an API. SAP API Management also provides policy-based functionality for configuring the endpoints that app developers can use to obtain access tokens.

Once you have configured OAuth policies on your API, apps must obtain access tokens to consume them. To do so, the app must exchange a request token for an access token. Rather than relying on a single password as the master key for every app that accesses an API, OAuth uses this token to provide “delegated authentication” between APIs and apps. The resource owner can issue access tokens with restricted scope and limited lifetime, and revoke them independently.

8.5.1.1 Configuring an OAuth 1.0a Policy

SAP API Management provides an OAuth 1.0a policy type that enables you to authorize apps with OAuth 1.0a. The OAuthV1 policy type is responsible for generating request tokens, generating access tokens, and verifying access tokens based on the OAuth 1.0a specification.

> **Note**
>
> The name attribute for this policy is restricted to these characters: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.

8.5.1.2 Generating a Request Token

A request token is used by a consumer app to obtain authorization from the app end user, and is then presented to the 'token endpoint' to be exchanged for an access token. A request token is generated from a valid consumer key. Here are examples of the simple and comprehensive forms for generating request tokens.

Simple form

```
<OAuthV1 name="OAuthV1-GenerateRequestToken-1">
  <Operation>GenerateRequestToken</Operation>
</OAuthV1>
```

Comprehensive form

```
<OAuthV1 name="OAuthV1-GenerateRequestToken-1">
  <Operation>GenerateRequestToken</Operation>
    <URL ref="flow.variable">value</URL>
    <GenerateResponse enabled="true">
    <Format>FORM_PARAM or XML</Format>
    </GenerateResponse>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <GenerateErrorResponse enabled="true">
      <Format>formparam or xml</Format>
      <Realm>http://oauth.mycompany.com/oauth/1/</Realm>
    </GenerateErrorResponse>
</OAuthV1>
```

The policy enforces the following OAuth semantics:

- The consumer key is valid.
- The signature is valid.

A successful request returns the following responses:

- Generates a request token and its attributes in the flow
- `oauth_token, oauth_token_secret,oauth_callback_confirmed, oauth_response`
- Also makes a consumer token and its attributes available in the flow
- `oauth_consumer_key,oauth_consumer_secret`

The <URL> element allows proper OAuthV1 signature calculation when running behind Elastic Load Balancers (ELBs). The ref attribute takes precedence when both the `ref` attribute and a text value is specified. If ref is not provided, or if the flow variable given in ref is NULL, then the text value will be considered for generating the signature.

For example:

```
<URL ref="secretURL">https://example-test.net/oauth/access_token</URL>
```

On failure, a request returns the appropriate response status code with an error message.

8.5.1.3 Generating an Access Token

An access token is used by the consumer to gain access to the protected resources on behalf of the user, instead of using the user's service provider credentials. An access token is created with a valid key, request token, and verifier combination. Here are examples of the simple and comprehensive forms for generating access tokens.

Simple form

```
<OAuthV1 name="OAuthV1-GenerateAccessToken-1">
   <Operation>GenerateAccessToken</Operation>
</OAuthV1>
```

Comprehensive form

```
<OAuthV1 name="OAuthV1-GenerateAccessToken-1">
  <Operation>GenerateAccessToken</Operation>
  <URL ref="flow.variable">{value}</URL>
  <GenerateResponse enabled="true">
    <Format>formparam or xml</Format>
  </GenerateResponse>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  <GenerateErrorResponse enabled="true">
    <Format>FORM_PARAM or XML</Format>
    <Realm>http://oauth.example.com/oauth/1/</Realm>
  </GenerateErrorResponse>
</OAuthV1>
```

The policy enforces the following OAuth semantics:

- The consumer key is valid.
- The request token is valid.
- The verifier is valid.
- The signature is valid.

A successful request returns the following responses:

- Generates an access token and its attributes in the flow
- `oauth_token, oauth_token_secret,oauth_response`
- Also makes a consumer token and its attributes available in the flow
- `oauth_consumer_key,oauth_consumer_secret`

The <URL> element allows proper OAuthV1 signature calculation when running behind Elastic Load Balancers (ELBs). The ref attribute takes precedence when both the ref attribute and a text value is specified. If `ref` is not provided, or if the flow variable given in `ref` is NULL, then the text value will be considered for generating the signature.

Example

<URL ref="secretURL">https://example-test.net/oauth/access_token</URL>

On failure, a request returns the appropriate response status code with an error message.

8.5.1.4 Associating a Token Verification Code with a Request Token

The following policy associates a token verification code with a request token. To receive an access token, both a verification code and a request token are required.

Note

To use this policy, a verification code must be generated as a separate step outside of SAP API Management. The verifier value must be a unique, random string.

The verification code fits into the basic OAuth flow as follows: After an app user's credentials are authenticated by a service provider, the service provider must inform the consumer app that the credentials were accepted, implying that the request token can be exchanged for an access token. This is usually done via a 302 redirect. Separately, the service provider must send a verification code to SAP API Management, and SAP API Management internally associates this verifier to the request token.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Later, when the app exchanges the request token for an access token, it must pass both the request token and the verifier code. SAP API Management checks that the verifier code is correct, ensuring that the user who authenticated and granted access to the service is the same user who is asking the app to retrieve an access token. For detailed information on this message flow, see the OAuth 1.0a specification.

Comprehensive form

```
<OAuthV1 name="OAuthV1-GenerateVerifier-1">
    <Operation>GenerateVerifier</Operation>
    <RequestToken ref="request.header.requesttoken"/>
    <AppUserId ref="request.header.appuserid"/>
    <VerifierCode ref="request.header.verifiercode"/>
    <GenerateResponse enabled="true">
       <Format>FORM_PARAM or XML</Format>
    </GenerateResponse>
    <Attributes>
       <Attribute name="ver-attr1">ver-attr1</Attribute>
       <Attribute name="ver-attr2" ref="request.header.ver-attr2"></Attribute>
       <Attribute name="ver-attr3" display="false">ver-attr3</Attribute>
    </Attributes>
</OAuthV1>
```

Parameters

Parameter	Description
RequestToken	(Required) Specifies the request token with which to associate the verifier.
AppUserId	(Required) Specifies the ID of the application user.
VerifierCode	(Required) Specifies the verifier code. This code must be generated outside of SAP API Management as a separate step.
Attributes	(Optional) Allows you to attach arbitrary values to the access token. It is also possible to attach attributes to the request token. Any attributes that are attach to the request token get "promoted" to the access token at the time the app exchanges the request token for the access token. If the policy specifies custom attributes to associate with the access token that have the same name as custom attributes already associated to the request token, the values of the custom attribute specified here take precedence. See also "Customizing access tokens" later in this topic.

When this policy executes, the following flow variables are populated:

- Request Token: `oauthtoken.{policy_name}.oauth_token`
- Verifier Code: `authtoken.{policy_name}.verifier_code`
- App User Id: `oauthtoken.{policy_name}.app_user_id`

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Sample response

If GenerateReponse is enabled, then the verifier is returned in the response along with the other attributes. This is an example response when Format is set to XML.

```
<response>
    <oauth_parameter name="issued_at">1389014375368</oauth_parameter>
    <oauth_parameter name="expires_at">1389016175368</oauth_parameter>
    <oauth_parameter name="application_name">f24b582d-701f-4cbd-a1e0-
2fc0e77003</oauth_parameter>
    <oauth_parameter name="app_user_id">someappuserid</oauth_parameter>
    <oauth_parameter name="organization_id">0</oauth_parameter>
    <oauth_parameter name="oauth_token">HVEUg3Tb9tM8yMWZAARDcSbgeHp5</oauth_parameter>
    <oauth_parameter name="ver-attr1">ver-attr1-value</oauth_parameter>
    <oauth_parameter name="oauth_verifier">verifier-code</oauth_parameter>
    <oauth_parameter name="verifier_code">verifier-code</oauth_parameter>
</response>
```

8.5.1.5 Verifying an Access Token

Here are examples of the simple and comprehensive forms for verifying access tokens.

Simple form

```
<OAuthV1 name="OAuthV1-VerifyAccessToken-1">
  <Operation>VerifyAccessToken</Operation>
</OAuthV1>
```

Comprehensive form

```
<OAuthV1 name="OAuthV1-VerifyAccessToken-1">
  <Operation>VerifyAccessToken</Operation>
  <URL ref="flow.variable">{value}</URL>
  <GenerateErrorResponse enabled="true">
    <Format>FORM_PARAM or XML</Format>
    <Realm>http://oauth.example-test.com/oauth/1/</Realm>
  </GenerateErrorResponse>
</OAuthV1>
```

The policy enforces the following OAuth semantics:

- The consumer key is valid.
- The access token is valid.
- The signature is valid.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

A successful validation returns the following responses:

- Makes an access token and its attributes available in the flow
- `oauth_token, oauth_token_secret,oauth_response`
- Also makes a consumer token and its attributes available in the flow
- `oauth_consumer_key,oauth_consumer_secret`

The <URL> element allows proper OAuthV1 signature calculation when running behind Elastic Load Balancers (ELBs). The ref attribute takes precedence when both the `ref` attribute and a text value is specified. If ref is not provided, or if the flow variable given in ref is NULL, then the text value will be considered for generating the signature.

If the request fails verification, the server responds with the appropriate response status code with an error message.

8.5.1.6 Customizing Access Tokens

Custom attributes can be optionally included in access, refresh, and verifier tokens.

```
<Attributes>
  <Attribute name=”attr_name1” ref=”flow.variable”
display="true|false">value1</Attribute>
  <Attribute name=”attr_name2” ref=”flow.variable”
display="true|false">value2</Attribute>
</Attributes>
```

Where the display is set to `true`, custom attributes are returned in the response, where they may be viewable by the app end user.

Where the display is set to `false`, custom attributes are stored in the data store, but are not returned in the response message.

For example, to add the User-Agent associated with the request to an access token:

```
<OAuthV1 name="GenerateAccessToken">
  <Operation>GenerateAccessToken</Operation>
  <ExpiresIn>1000</ExpiresIn>
  <GenerateResponse />
  <Attributes>
    <Attribute name=”user-agent” ref=”request.header.user-agent”
            display="false">NONE</Attribute>
  </Attributes>
</OAuthV1>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.1.7 Policy-specific Variables

The OAuthV1 and GetOAuthV1Info Policy types set the following variables when they execute.

GenerateRequestToken operation

- oauth_token
- oauth_token_secret
- oauth_callback_confirmed
- oauth_response
- oauth_consumer_key
- oauth_consumer_secret

GenerateAccessToken operation

- oauth_token
- oauth_token_secret
- oauth_response
- oauth_consumer_key
- oauth_consumer_secret

VerifyAccessToken operation

- oauth_token
- oauth_token_secret
- oauth_response
- oauth_consumer_key
- oauth_consumer_secret

GetOAuthV1Info

- oauth_consumer_key
- oauth_consumer_secret

8.5.1.8 Policy-specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

The OAuthV1 Policy type defines the following error codes:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Error Code	Message
AppKeyNotResolved	Could not resolve the app key with variable{0}
ConsumerKeyNotResolved	Could not resolve the consumer key withvariable {0}
RequestTokenNotResolved	Could not resolve the request token with the variable {0}
AccessTokenNotResolved	Could not resolve the access token with the variable {0}
ResponseGenerationError	Error while generating response : {0}
UnableToDetermineOperation	Unable to determine an operation forstepDefinition {0}
UnableToResolveOAuthConfig	Unable to resolve the OAuth configuration for{0}
AtLeastOneParamRequired	At least one of AccessToken, RequestToken orConsumerKey must be specified instepDefinition {0}
SpecifyValueOrRefReqToken	Specify Request Token as value or ref instepDefinition {0}
SpecifyValueOrRefAccToken	Specify Access Token as value or ref instepDefinition {0}
SpecifyValueOrRefConKey	Specify Consumer Key as value or ref instepDefinition {0}
SpecifyValueOrRefAppKey	Specify App Key as value or ref instepDefinition {0}
ExpiresInNotApplicableForOperation	ExpiresIn element is not valid for operation{0}
InvalidValueForExpiresIn	Invalid value for ExpiresIn element foroperation {0}
FailedToFetchApiProduct	Failed to fetch api product for key {0}
InvalidTokenType	Valid token types : {0}, Invalid toke type {1}in stepDefinition {2}
TokenValueRequired	Token value is required in stepDefinition {0}
FailedToResolveRealm	Failed to resolve realm {0}

8.5.2 Authorize Requests Using OAuth 2.0

OAuth 2.0 defines an authorization protocol for protected API resources. To ensure that apps are allowed to act on behalf of users, OAuth 2.0 relies on 'access tokens'. To access protected resources, consumer apps must obtain 'access tokens'. The OAuth 2.0 specification defines the various ways that apps can request and use access tokens. SAP API Management provides a policy type that enables you to configure OAuth 2.0 authorization for your APIs.

For step-by-step instructions on using the default OAuth 2.0 endpoints provided by SAP API Management see Secure APIs with OAuth 2.0: client credentials.

Setting up OAuth 2.0 authorization for your API is a three step process:

1. Configure a token endpoint: An OAuth token endpoint defines a URI on SAP API Management. The token endpoint is configured with a policy of type `OAuthV2`. In the `OAuthV2` policy, the GenerateAccessToken operation is specified. When this operation is specified, you have the option of configuring one or more grant

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

types. For each grant type specified, an additional set of configuration elements are exposed, providing flexibility in the way that APIs exposed through SAP API Management manage OAuth-based authorization.

2. Apply an OAuth validation policy to protected resource URIs: To enforce OAuth at runtime, attach a policy of type `OAuthV2` to a Flow that exposes a protected resource. In the OAuthV2 policy, specify the `VerifyAccessToken` operation.
3. Configure one or more API products: The VerifyAccessToken operation resolves the access token to an API product for which the app has been approved. The request URI is verified against the list of URIs defined in the API product. If the request URI is included in the list defined by the approved API product, then the request is forwarded to the protected resource.

> **Note**
>
> Default OAuth 2.0 client credentials token and refresh endpoints are provisioned for every organization. Also, a default API product is configured for every API proxy that you create in your organization.
> See OAuth.

8.5.2.1 Configuring token endpoints

OAuth 2.0 policies are used both to generate and to validate OAuth 2.0-compliant tokens. To generate tokens on behalf of app end users, OAuth 2.0 policies that specify the GenerateAccessToken operation are attached to a token endpoint.

> **Note**
>
> Deploy a single API proxy to function as a token endpoint for all API proxies in an environment. A single API proxy configured as a token endpoint can support multiple grant types. By setting up a single token endpoint, you can publish a unified set of URIs that app developers can use to obtain tokens.

A token endpoint is simply a URI path that the system uses to identify requests for access tokens. On SAP API Management, a token endpoint is a conditional Flow to which an OAuthV2 policy is attached. The OAuthV2 policy specifies the GenerateAccessToken operation as an element.

For example, to configure a token endpoint that generates tokens on requests to the URI path /accesstoken:

```
<Flow name="TokenEndpoint">
  <Condition>proxy.pathsuffix MatchesPath "/accesstoken"</Condition>
  <Request>
    <Step><Name>GenerateAccessToken</Name></Step>
  </Request>
</Flow>
```

> **Note**
>
> The variable proxy.pathsuffix defines the URI fragment following the base path configured in the ProxyEndpoint.
>
> `http://<host_name>:<port>/oauth/accesstoken`

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The token endpoint URL is provided to your app developers, who invoke this URL in their apps to obtain access tokens at runtime.

8.5.2.2 Configuring Authorization Endpoints

Similarly, you can configure authorization endpoints to issue authorization codes. For example:

```
<Flow name="AuthorizationEndpoint">
  <Condition>proxy.pathsuffix == "/authorize"</Condition>
  <Request>
    <Step><Name>GenerateAuthCode</Name></Step>
  </Request>
</Flow>
```

For instructions on configuring an API proxy to support authorization code, please see Secure APIs with OAuth 2.0: client credentials.

OAuth 2.0 Grant Types

OAuth 2.0 policies generate access tokens using the GrantType method element. Four grant types, specified by OAuth 2.0, are supported:

Grant Type	Description
Clientcredentials	"Two-legged" OAuth, usually implemented for trusted clients (for example, apps developed by the API provider themselves).
Authorizationcode	"Three-legged" OAuth, which enables an app end users to obtain an access token without exposing credentials to the app. The app requests an access token using an authorization code returned by the intermediary who authenticates the app end user. API Platform can act as both authorization server (generating authorization codes) and as a token endpoint (issuing access tokens in return for valid authorization codes).
Implicit	A variation on authorization code, usually enforced for browser-based apps that are implemented in scripting languages such as JavaScript
ResourceOwnerPasswordCredentials	Used only when apps are trusted. Enables an app end user to obtain an access token in return for a valid username/password credentials. The benefit of password grant type is the one-time use of password credentials to obtain a long-lived access token.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.2.3 Verifying Access Tokens

Once a token endpoint is set up for an API proxy, a corresponding OAuthV2 policy that specifies theVerifyAccessToken operation is attached to the Flow that exposes the protected resource.

Example

To ensure that all requests to an API are authorized, the following policy enforces access token verification:

```
<OAuthV2 name="VerifyOAuthAccessToken">
  <Operation>VerifyAccessToken</Operation>
</OAuthV2>
```

The policy is attached to the API resource to be protected. To ensure that all requests to an API are verified, attach the policy to the ProxyEndpoint request PreFlow, as follows:

```
<PreFlow>
  <Request>
      <Step><Name>VerifyOAuthAccessToken</Name></Step>
  </Request>
</PreFlow>
```

The following optional elements can be used to override the default settings for the VerifyAccessToken operation.

Name	Description
Scope	A space separated list of scopes that must be present in the access token for verification to succeed. For example, the following policy will check the access token to ensure that it contains the scopes READ and WRITE: <OAuthV2 name="ValidateOauthScopePolicy"> <Operation>VerifyAccessToken</Operation> <Scope>READ WRITE</Scope> </OAuthV2>
AccessToken	The variable where the access token is expected to be located. For examplerequest.queryparam.accesstoken. By default, the access token is expected to be presented by the app in the Authorization HTTP header, according to the OAuth 2.0 specification. Use this setting if the access token is expected to be presented in a non-standard location, such as a query parameter, or an HTTP header with a name other than Authorization.

For instructions on obtaining and verifying tokens, see OAuth.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.2.4 Grant Type Policy Details

The OAuth 2.0 policy enables the user to specify which API calls to secure with OAuth 2.0. The OAuth policy exposes configuration variables that enable the user to configure the behavior of the policy. SAP API Management supports the following OAuth policies based on the OAuth 2.0 specification.

Grant Type	Policy Format	Description	Configuration Example

authorization_code	Generate Authorization Code	Generates an authorization code that can in turn be used to get an access token. Attach the policy at the authorization end point.	`<OAuthV2 name="GetAuthCode">`
			
			`<Operation>GenerateAuthorizationCode</Operation>`
			`<ExpiresIn>1000</ExpiresIn>`
			`<GenerateResponse />`
			`</OAuthV2>`

	Generate AccessToken	Generates an access token in exchange for an authorization code. The access token can be used to gain access to the protected resources (target APIs). Attach the policy at the access token endpoint.	`<OAuthV2 name="GenerateAccessToken">`
			`<Operation>GenerateAccessToken</Operation>`
			`<ExpiresIn>1000</ExpiresIn>`
			`<GenerateResponse />`
			`<SupportedGrantTypes>`
			`<GrantType>authorization_code</GrantType>`
			`</SupportedGrantTypes>`
			
			`<GrantType>request.queryparam.grant_type</GrantType>`
			`<Code>request.queryparam.code</Code>`
			
			`<ClientId>request.queryparam.client_id</ClientId>`
			
			`<RedirectUri>request.queryparam.redirect_uri</RedirectUri>`
			`<Scope>request.queryparam.scope</Scope>`
			`</OAuthV2>`

	Refresh AccessToken	Token used to re-issue the access	`<OAuthV2 name="RefreshAccessToken">`
			`<Operation>RefreshAccessToken</Operation>`
			`<ExpiresIn>1000</ExpiresIn>`

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Grant Type	Policy Format	Description	Configuration Example
		token post expiry. Attach the policy at the refresh token end point.	<GenerateResponse /> <GrantType>request.queryparam.grant_type</GrantType> </OAuthV2>
implicit	GenerateImplicitAccessToken	Attach the policy at the access token endpoint.	<OAuthV2 name="GenerateAccessTokenImplicit"> <Operation>GenerateAccessTokenImplicitGrant</Operation> <ExpiresIn>1000</ExpiresIn> <GenerateResponse /> <ResponseType>request.queryparam.response_type</ResponseType> <ClientId>request.queryparam.client_id</ClientId> <RedirectUri>request.queryparam.redirect_uri</RedirectUri> <Scope>request.queryparam.scope</Scope> <State>request.queryparam.state</State> <AppEndUser>request.queryparam.app_enduser</AppEndUser> </OAuthV2>
password	GenerateAccessToken	Attach the policy at the access token endpoint.	<OAuthV2 name="GenerateAccessToken"> <Operation>GenerateAccessToken</Operation> <ExpiresIn>1000</ExpiresIn> <GenerateResponse /> <SupportedGrantTypes> <GrantType>password</GrantType> </SupportedGrantTypes> <GrantType>request.queryparam.grant_type</GrantType> <ClientId>request.queryparam.client_id</ClientId> <Scope>request.queryparam.scope</Scope> <AppEndUser>request.queryparam.app_enduser</AppEndUser> <UserName>request.queryparam.user_name</UserName> <PassWord>request.queryparam.password</PassWord> </OAuthV2>

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Grant Type	Policy Format	Description	Configuration Example

	Refresh AccessToken	Attach the policy at the refresh token end point.	`<OAuthV2 name="RefreshAccessToken">`
			`<Operation>RefreshAccessToken</Operation>`
			`<ExpiresIn>1000</ExpiresIn>`
			`<GenerateResponse />`
			
			`<GrantType>request.queryparam.grant_type</GrantType>`
			`</OAuthV2>`

client_credentials	Generate AccessToken	Attach the policy at the access token end point.	`<OAuthV2 name="GenerateAccessToken">`
			`<Operation>GenerateAccessToken`
			`</Operation>`
			`<ExpiresIn>1000</ExpiresIn>`
			`<GenerateResponse />`
			`<SupportedGrantTypes>`
			`<GrantType>client_credentials</GrantType>`
			`</SupportedGrantTypes>`
			`<GrantType>request.queryparam.grant_type</GrantType>`
			`</OAuthV2>`

Note

The ExpiresIn element (optional) enforces the expiry time of the authorization code in milliseconds. The expiry time of authorization code is system generated plus ExpiresIn value. If ExpiresIn is set to -1, the system considers it as an infinite life time. If ExpiresIn is not specified, the system applies a default value configured at the system level.

8.5.2.5 OAuth 2.0 Policy Settings Reference

Name	Description	Valid Values	Related Operation and Grant Type Combinations
name	The name of the policy. Must be unique within the API proxy configuration	Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.	All
Operation	The OAuth 2.0 operation	GenerateAccessToken GenerateAccessTokenImplicitGrant	All

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Valid Values	Related Operation and Grant Type Combinations
	implemented by the policy	GenerateAuthorizationCode RefreshAccessToken VerifyAccessToken	
ExpiresIn	ExpiresIn (optional) enforces the expiry time of the authorization code in milliseconds. The expiry time of authorization code is system generated plusExpiresInvalue. IfExpiresInis -1, the system considers it as a infinite life time. If it is not specified, the system applies a default value configured at system level.	Any integer, including -1 (which indicates that access tokens do not expire). The expiration can also be set using a reference to a variable, as follows. <ExpiresIn ref="flow.variable">{default_value}</ExpiresIn>	All
ReuseRefresh Token	Indicates whether a new RefreshToken should be issued when a valid RefreshToken is presented. When set to true, the existing RefreshToken is reused until it expires.	true or false	All except client credentials and implicit
RefreshToken ExpiresIn	Enforces the expiry time of refresh tokens in milliseconds.	Any integer, including -1 (which indicates that refresh tokens do not expire) The expiration can also be set using a reference to a variable, as follows. <RefreshTokenExpiresIn ref="flow.variable">{default_value}</RefreshTokenExpiresIn>	All except client credentials and implicit

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Valid Values	Related Operation and Grant Type Combinations
SupportedGrantTypes	Specifies the GrantTypes supported by a token endpoint. An endpoint may support multiple GrantTypes (that is, a single endpoint can be configured to distribute access tokens for multiple GrantTypes.)	client_credentials authorization_code password implicit	All
Code	The expected location in the request message where the authorization code must be presented to the token endpoint.	Any variable setting. For example request.queryparam.auth_code indicates that the authorization code should be present as a query parameter, as, for example, ?auth_code=AfGlvs9. To require the authorization code in an HTTP header, for example, set this value to request.header.auth_code.	GenerateAccessToken with grant type authorization_code
ClientId	The expected location in the request message where the client_id (the app's consumer key) must be presented to the token endpoint.	Any variable setting. For example request.queryparam.client_id indicates that the client_id should be present as a query parameter, as, for example, ?client_id=AfGlvs9. To require the ClientId in an HTTP header, for example, set this value to request.header.client_id.	GenerateAccessToken Implicit: Optional GenerateAuthorization Code:Optional
RedirectUri	The expected location in the request message where the RedirectUri must be presented to the token endpoint.	Any variable setting. For example request.queryparam.redirect_uriindicates that the RedirectUri should be present as a query parameter, as, for example,? redirect_uri=login.myapp.com. To require the RedirectUri in an HTTP header, for example, set this value to request.header.redirect_uri.	GenerateAccessToken Implicit: Optional GenerateAuthorization Code:Optional
Scope	The expected location in the request message where the scope must be	Any variable setting. For example request.queryparam.scope indicates that the scope should be present as a query parameter, as, for example, ?scope=READ. To require the scope in an HTTP header, for	All: Optional

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Valid Values	Related Operation and Grant Type Combinations
	presented to the token endpoint.	example, set this value to request.header.scope.	
State	The expected location in the request message where the state must be presented to the token endpoint.	Any variable setting. For example request.queryparam.state indicates that the state should be present as a query parameter, as, for example, ?state=HjoiuKJH32. To require the state in an HTTP header, for example, set this value to request.header.state.	authorization_code, password
AppEndUser	The expected location in the request message where the state must be presented to the token endpoint.	Any variable setting. For example request.queryparam.app_enduserindicates that the AppEndUser should be present as a query parameter, as, for example,?app_enduser=ntesla@theramin.com. To require the AppEndUser in an HTTP header, for example, set this value torequest.header.app_enduser.	All: Optional
UserName	The expected location in the request message where the UserName must be presented to the token endpoint.	Any variable setting. For example request.queryparam.username indicates that the username should be present as a query parameter, as, for example,?username=joe. To require the UserName in an HTTP header, for example, set this value to request.header.username.	All: Optional
Password	The expected location in the request message where the Password must be presented to the token endpoint.	Any variable setting. For example request.queryparam.password indicates that the Password should be present as a query parameter, as, for example,?password=changeit. To require the Password in an HTTP header, for example, set this value to request.header.password.	All: Optional
GenerateResponse	An element used in token endpoints, authorization endpoints, and refresh endpoints to indicate that a response should be generated for requests, and that	N/A	All: Optional

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Valid Values	Related Operation and Grant Type Combinations
	no further processing should take place. (Indicates that the policy is an endpoint.)		

8.5.2.6 Customizing Access Tokens

Custom attributes can be optionally included in access tokens.

```
<Attributes>
  <Attribute name="attr_name1" ref="flow.variable"
display="true|false">value1</Attribute>
  <Attribute name="attr_name2" ref="flow.variable"
display="true|false">value2</Attribute>
</Attributes>
```

Where display is set to true, custom attributes are returned in the response, where they may be viewable by the app end user.

Where display is set to false, custom attributes are stored in the data store, but are not returned in the response message.

For example, to add the User-Agent associated with the request to an access token:

```
<OAuthV2 name="GenerateAccessToken">
  <Operation>GenerateAccessToken
  </Operation>
  <ExpiresIn>1000</ExpiresIn>
  <GenerateResponse />
  <SupportedGrantTypes>
    <GrantType>client_credentials</GrantType>
  </SupportedGrantTypes>
  <GrantType>request.queryparam.grant_type</GrantType>
  <Attributes>
    <Attribute name="user-agent" ref="request.header.user-agent"
display="false"></Attribute>
  </Attributes>
</OAuthV2>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.2.7 Delegating Token Management

Remote token stores can be used against SAP API Management. If a remote token store is used, the following element should be configured in the GenerateAccessToken policy. When this configuration is used, SAP API Management will generate the access token and return it to the requesting app.

```
<StoreToken>false</StoreToken>
```

If token validation has been delegated to a remote token service, then the following element must be configured, identifying the location of the token in the request message. By specifying an ExternalAccessToken element, you indicate that SAP API Management should not attempt to validate the token.

```
<ExternalAccessToken> request.header.external_access_token |
request.formparam.external_access_token | request.queryparam.external_access_token
</ExternalAccessToken>
```

Note that if you configure an external access token service, you must implement a ServiceCallout that enforces the token check.

> **Note**
>
> Only bearer tokens are currently supported. MAC tokens are not supported. The access token must be passed in the authorization header. For example, a sample authorization header is:
>
> ```
> "Authorization: Bearer {access_token}"
> ```

8.5.2.8 Policy-specific Variables

OAuth policies populate a large set of default variables.

For a comprehensive list, see OAuth flow variables.

8.5.2.9 Invalidating Tokens

In some cases, apps are required to explicitly revoke or invalidate tokens, for example, when a user logs out of an OAuth-enabled app.

The procedure for token revocation is defined by the IETF Token Revocation draft specification.

The specification states:

[T]oken revocation prevents abuse of abandoned tokens and facilitates a better end-user experience since invalidated authorization grants will no longer turn up in a list of authorization grants the authorization server might present to the end-user.

SP API Management provides an InvalidateToken operation that enables you to configure a dedicated token revocation endpoint. By publishing the URI of this endpoint, you enable app developers to invalidate tokens issued by SAP API Management.

For example:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<OAuthV2 name="InvalidateToken">
  <Operation>InvalidateToken</Operation>
  <Tokens>
    <Token type="accesstoken" cascade="true">flow.variable</Token>
    <Token type="refreshtoken" cascade="true">flow.variable</Token>
  </Tokens>
</OAuthV2>
```

Configure the InvalidateToken operation using the following elements.

Field Name	Descripti
Tokens	Identifies the Flow variable that defines the source of the token to be revoked. If developers are expected to submit access tokens as query parameters named access_token, for example, userequest.queryparam.access_token. · type (Optional): The token type identified by the variable specified. Supported values areaccesstoken and refreshtoken. · cascade (Optional): A boolean that determines whether the revocation action will propagate to associated tokens. Supported values are true and false. Defaults to true if not configured.

You can use the same technique to approve access tokens. The OAuthV2 policy type supports a ValidateToken operation. The settings are the same as the InvalidateToken operation. The difference is that, by specifying the ValidateToken operation, the status of the targeted access token or refresh token from 'revoked' to 'approved'.

```
<OAuthV2 name="ValidateToken">
  <Operation>ValidateToken</Operation>
  <Tokens>
    <Token type="accesstoken" cascade="true">flow.variable</Token>
    <Token type="refreshtoken" cascade="true">flow.variable</Token>
  </Tokens>
</OAuthV2>
```

8.5.2.10 Policy-specific error codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

The OAuthV2 Policy type defines the following error codes:

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

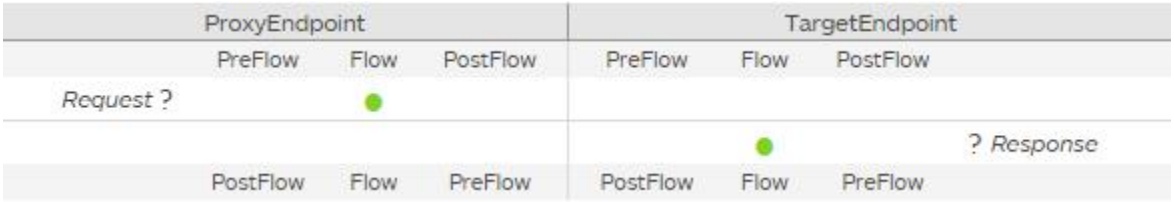
Error Code	Message
FailedToResolveOAuthConfig	Failed to resolve oauth config {0}
OperationRequired	Operation required
InvalidOperation	Operation {0} is invalid
InsufficientScope	Required scope(s) : {0}
FailedToResolveClientId	Failed to resolve client id variable {0}
FailedToResolveAccessToken	Failed to resolve access token variable {0}
FailedToResolveRefreshToken	Failed to resolve refresh token variable {0}
FailedToResolveToken	Failed to resolve token variable {0}
FailedToResolveAuthorizationCode	Failed to resolve authorization code variable{0}
ExpiresInNotApplicableForOperation	ExpiresIn element is notvalid for operation {0}
RefreshTokenExpiresInNotApplicableForOperation	RefreshToken ExpiresInelement is not valid foroperation {0}
InvalidValueForExpiresIn	Invalid value forExpiresIn element foroperation {0}
InvalidValueForRefreshTokenExpiresIn	Invalid value for RefreshToken ExpiresIn element foroperation {0}
InvalidGrantType	Invalid grant type: {0}
GrantTypesNotApplicableForOperation	Grant types are notapplicable for operation{0}
UnSupportedGrantType	Unsupported Grant Type:{0}
InvalidParameter	Invalid required paramater: {0}
MissingParameter	Missing required paramater: {0}
SpecifyValueOrRefApiKey	Specify Api Key as valueor ref in stepDefinition{0}
SpecifyAPIProduct	Specify the API Product toget details {0}
FailedToFetchApiProduct	Failed to fetch api product for key {0}
FailedToResolveAPIKey	Failed to resolve API Keyvariable {0}
FailedToResolveAPIProduct	Failed to resolve API product variable {0}
InvalidTokenType	Valid token types : {0},Invalid toke type {1} instepDefinition {2}
TokenValueRequired	Token value is required instepDefinition {0}
FailedToResolveRealm	Failed to resolve realm{0}

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.3 Authorize Requests using OAuthV2 Policy

OAuthV2 is a multi-faceted policy for performing OAuth 2.0 grant type operations. This is the primary policy used to configure OAuth 2.0 endpoints.

Place this policy on custom API proxy flows to generate access tokens, refresh tokens, authorization codes, to perform token validation, and other operations. For example, you might put this policy on an API proxy endpoint called /authorizationcode for apps to request authorization codes. Likewise, you might put this policy on an API proxy endpoint called /token to generate an access token when that endpoint is called.



8.5.3.1 Verifying Access Tokens

This OAuthV2 policy configuration (with the VerifyAccessToken operation) verifies that an access token submitted to SAP API Management is valid. When this policy operation is triggered, SAP API Management looks for a valid access token in the request. If the access token is valid, the request is allowed to proceed. If invalid, all processing stops and an error is returned in the response.

```
<OAuthV2 async="false" continueOnError="false" enabled="true" name="OAuth-v20-2">
    <DisplayName>OAuth v2.0 2</DisplayName>
    <Operation>VerifyAccessToken</Operation>
    <AccessTokenPrefix>Bearer</AccessTokenPrefix> <!-- Optional, default is Bearer -->
</OAuthV2>
```

Note that only bearer (access) tokens are supported. MAC tokens are not supported.

Apps must pass the access token in the Authorization HTTP request header. For example:

```
$ curl -H "Authorization: Bearer ylSkZIjbdWybfsUQe9BqP0LH5Z" http://{org-name}-
{env}.<host:port>/weather/forecastrss?w=12797282
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.3.2 Generating Access Tokens

For examples showing how to request access tokens for each of the supported grant types, see Requesting access tokens and authorization codes. The topic includes examples of these operations:

- Authorization code
- Client credentials
- Implicit
- Password
- Refresh Token
- Authorization code

8.5.3.3 Generating Authorization Code

For examples showing how to request authorization codes, see Requesting access tokens and authorization codes.

8.5.3.4 Refreshing an Access Token

For examples showing how to request access tokens using a refresh token, see Requesting access tokens and authorization codes.

8.5.3.5 Generating Access Token on the Response Flow

Sometimes you may need to generate an access token in the response flow. For example, you may do this in response to some custom validation done in a backend service. In this example, the use case requires both an access token and a refresh token, ruling out the implicit grant type. In this case, we'll use the password grant type to generate the token. As you'll see, the trick to making this work is to pass in an Authorization request header with a JavaScript policy.

First, let's look at the sample policy:

```
<OAuthV2 enabled="true" continueOnError="false" async="false"
name="generateAccessToken">
    <Operation>GenerateAccessToken</Operation>
    <AppEndUser>Doe</AppEndUser>
    <UserName>jdoe</UserName>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <PassWord>jdoe</PassWord>
    <GrantType>grant_type</GrantType>
    <ClientId>a_valid_client_id</ClientId>
    <SupportedGrantTypes>
        <GrantType>password</GrantType>
    </SupportedGrantTypes>
</OAuthV2>
```

If you put this policy on the response flow, it will fail with a 401 UnAuthorized error even though the correct login parameters are specified in the policy. To solve this problem, you need to set an Authorization request header.

The Authorization header must contain a Basic access scheme with the Base64-encoded client_id:client_secret.

You can add this header with a JavaScript policy placed just before the OAuthV2 policy, like this. The "local_clientid" and "local_secret" variables must be previously set and available in the flow:

```
var client_id = context.getVariable("local_clientid");
var client_secret = context.getVariable("local_secret");
context.setVariable("request.header.Authorization","Basic
"+CryptoJS.enc.Base64.stringify(CryptoJS.enc.Latin1
                                    .parse(client_id + ':' + client_secret)));
```

8.5.3.6 Element Reference

> **Note**
>
> The way you configure this policy, and the elements you need to specify, depend on which operation you want the policy to perform. For example, if you are implementing the authorization code grant type, then you will require four separate OAuthV2 policies to perform authorization code generation, access code generation, access code validation, and refresh token generation. This reference lists all of the elements that can be used with this policy.

The sample policy shown below is one of many possible configurations. This sample shows an OAuthV2 policy configured for the GenerateAccessToken operation. It includes required and optional elements. Refer to the element descriptions in this section for details.

```
<OAuthV2 name="GenerateAccessToken">
  <Operation>GenerateAccessToken</Operation>
  <ExpiresIn>1000</ExpiresIn>
  <GenerateResponse />
  <SupportedGrantTypes>
   <GrantType>authorization_code</GrantType>
  </SupportedGrantTypes>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  <GrantType>request.queryparam.grant_type</GrantType>
  <Code>request.queryparam.code</Code>
  <ClientId>request.queryparam.client_id</ClientId>
  <RedirectUri>request.queryparam.redirect_uri</RedirectUri>
  <Scope>request.queryparam.scope</Scope>
</OAuthV2>
```

8.5.3.7 <OAuthV2> attributes

```
<OAuthV2 async="false" continueOnError="false" enabled="true" name="MyOAuthPolicy">
```

The following attributes are common to all policy parent elements.

Attribute	Description	Default	Presence
name	The internal name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the <DisplayName> element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Required
continueOnError	Set to false to return an error when a policy fails. This is expected behavior for most policies. Set to true to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to true to enforce the policy. Set to false to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to true, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to true improves API proxy performance. However, overusing async can hurt performance with too much thread switching.	false	Optional

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.3.8 <DisplayName> element

Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

<DisplayName>Policy Display Name</DisplayName>

Default:	N/A If you omit this element, the the value of the policy'sname attribute is used.
Presence:	Optional
Type:	String

8.5.3.9 <AccessToken> element

```
<AccessToken>request.header.access_token</AccessToken>
```

By default, VerifyAccessToken expects the access token to be sent in an Authorization header. You can change that default using this element. For example request.queryparam.access_token indicates that the access token should be present as a query parameter.

If presented in the header, it must be an Authorization header and be sent as a Bearer token. For example:

```
-H "Authorization: Bearer Rft3dqrs56Blirls56a"
```

Currently, Bearer is the only supported prefix.

Default:	request.header.access_token
Presence:	Optional
Type:	String
Valid values:	A valid access token
Used with operations:	VerifyAccessToken

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.3.10 <AccessTokenPrefix> element

```
<AccessTokenPrefix>Bearer</AccessTokenPrefix>
```

By default, VerifyAccessToken expects the access token to be sent in an Authorization header as a Bearer token. For example:

```
-H "Authorization: Bearer Rft3dqrs56Blirls56a"
```

Currently, Bearer is the only supported prefix.

Default:	Bearer
Presence:	Optional
Type:	String
Valid values:	Bearer
Used with operations:	VerifyAccessToken

8.5.3.11 <AppEndUser> element

```
<AppEndUser>request.queryparam.app_enduser</AppEndUser>
```

In cases where the app end user ID must be sent to the authorization server, this element lets you specify where SAP API Management should look for the end user ID. For example, it could be sent as a query parameter or in an HTTP header.

For example request.queryparam.app_enduser indicates that the AppEndUser should be present as a query parameter, as, for example, ?app_enduser=ntesla@theramin.com. To require the AppEndUser in an HTTP header, for example, set this value to request.header.app_enduser.

Providing this setting enables you to include an app end user ID in the access token. This feature is useful if you want to be able to retrieve or revoke OAuth 2.0 access tokens by end user ID.

Default:	N/A

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Presence:	Optional
Type:	String
Valid values:	Any flow variable accessible to the policy at runtime.
Used with grant types:	authorization_code implicit password client_credentials

8.5.3.12 <Attributes/Attribute>

```
<Attributes>
    <Attribute name=”attr_name1” ref=”flow.variable”
display="true|false">value1</Attribute>
    <Attribute name=”attr_name2” ref=”flow.variable”
display="true|false">value2</Attribute>
</Attributes>
```

Use this element to add custom attributes to an access token or authorization code. For example, you may wish to embed a user ID or session identifier in an access token that can be extracted and checked at runtime.

This element takes a value from a flow variable or a default value which is specified in the policy. If both are specified, the value specified in the flow variable is taken. The optional display attribute can be set to true to false. If set to true, the attribute will not be shown in the response. Default value is false.

Default:	N/A
Presence:	Optional
Valid values:	name -The attribute name ref - The value of the attribute. Can come from a flow variable. display - (Optional) If set to true, the attribute will not be shown in the response. Default value is false.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Used with grant types:	authorization_code implicit password client_credentials Can also be used with the GenerateAuthorizationCode operation.

8.5.3.13 <ClientId> element

```
<ClientId>request.queryparam.client_id</ClientId>
```

In several cases, the client app must send the client ID to the authorization server. This element lets you specify where SAP API Management should look for the client ID. For example, it could be sent as a query parameter or in an HTTP header.

The variable request.queryparam.client_id indicates that the client_id should be present as a query parameter, as, for example, ?client_id=AfGlvs9. To require the ClientId in an HTTP header, for example, set this value to request.header.client_id.

Default:	request.formparam.client_id (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Any flow variable accessible to the policy at runtime
Used with grant types:	authorization_code password implicit client_credentials Can also be used with the GenerateAuthorizationCode operation.

8.5.3.14 <Code> element

```
<Code>request.queryparam.code</Code>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

In the authorization grant type flow, the client must submit an authorization code to the authorization server. This element lets you specify where SAP API Managememnt should look for the authorization code. For example, it could be sent as a query parameter, HTTP header, or form parameter (the default).

The variable, request.queryparam.auth_code indicates that the authorization code should be present as a query parameter, as, for example, ?auth_code=AfGlvs9. To require the authorization code in an HTTP header, for example, set this value to request.header.auth_code.

Default:	request.formparam.code (a x-www-form-urlencoded and specified in the request body)
Presence:	optional
Type:	String
Valid values:	Any flow variable accessible to the policy at runtime
Used with grant types:	authorization_code

8.5.3.15 <ExpiresIn> element

```
<ExpiresIn>10000</ExpiresIn>
```

Enforces the expiry time of access tokens, refresh tokens, and authorization codes in milliseconds. The expiry time value is a system generated value plus the <ExpiresIn> value. If <ExpiresIn> is set to -1, the token or code is given an infinite lifetime. If <ExpiresIn> is not specified, the system applies a default value configured at the system level.

The expiry time can also be set at runtime using a reference to a flow variable. The flow variable can be retrieved from a header, query parameter, or form parameter (default). Or, it can be a hard-coded value.

For example request.queryparam.expires_in indicates that the expiry value should be present as a query parameter, as, for example, ?expires_in=360000. To require the value to come from an HTTP header, for example, set this value to request.header.expires_in.

The following stanza specifies a flow variable and a default value as well. Note that the flow variable value takes precedence over the specified default value.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<ExpiresIn ref="flow.variable">
    {default_value}
</ExpiresIn>
```

Default:	If not specified, the system applies a default value configured at the system level.
Presence:	Optional
Type:	Integer
Valid values:	Any integer, including -1 (which indicates an infinite expiry time).
Used with grant types:	authorization_code implicit password client_credentials refresh_token Also used with GenerateAuthorizationCode operation.

8.5.3.16 <ExternalAccessToken> element

```
<ExternalAccessToken>request.queryparam.external_access_token</ExternalAccessToken>
```

Tells SAP API Management where to find an external access token (an access token not generated by SAP API Management).

The variable request.queryparam.external_access_token indicates that the external access token should be present as a query parameter, as, for example, ?external_access_token=12345678. To require the external access token in an HTTP header, for example, set this value to request.header.external_access_token.

8.5.3.17 <ExternalAuthorization> element

```
<ExternalAuthorization>true</ExternalAuthorization>
```

If this element is false or not present, then SAP API Management validates the client_id and client_secret normally against the SAP API Management authorization store. Use this element when you want to work with third-party OAuth tokens.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Default:	false
Presence:	Optional
Type:	Boolean
Valid values:	true or false
Used with grant types:	authorization_code password client_credentials

8.5.3.18 <GenerateResponse> element

```
<GenerateResponse enabled='true'/>
```

If set to true, the policy generates and returns a response. If false, no response is sent. Instead, a set of flow variables are populated with values related to the policy's function. For example, a flow variable called oauthv2authcode.OAuthV2-GenerateAuthorizationCode.code gets populated with the newly minted auth code.

Default:	false
Presence:	Optional
Type:	string
Valid values:	true or false
Used with grant types:	implicit password client_credentials refresh_token Also can be used with the GenerateAuthorizationCode operation.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.3.19 <GenerateErrorResponse> element

```
<GenerateErrorResponse enabled='true'/>
```

If set to true, the policy generates and returns a response if the ContinueOnError attribute is set to true.
If false (the default), no response is sent. Instead, a set of flow variables are populated with values related to the policy's function.

Default:	false
Presence:	Optional
Type:	string
Valid values:	true or false
Used with grant types:	implicit password client_credentials refresh_token Also can be used with the GenerateAuthorizationCode operation.

8.5.3.20 <GrantType>

```
<GrantType>request.queryparam.grant_type</GrantType>
```

Tells the policy where to find the grant type parameter that is passed in a request. Per the OAuth 2.0 specification, the grant type must be supplied with requests for access tokens and authorization codes. The variable can be a header, query parameter, or form parameter (default). Or, it can be a hard-coded value.

For example request.queryparam.grant_type indicates that the Password should be present as a query parameter, as, for example, ?grant_type=password. To require the grant type in an HTTP header, for example, set this value to request.header.grant_type.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Default:	request.formparam.grant_type (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	string
Valid values:	A variable, as explained above, or a hard-coded grant type of: client_credentials, implicit, authorization_code, or password
Used with grant types:	authorization_code password implicit client_credentials refresh_token

8.5.3.21 <Operation> element

```
<Operation>GenerateAuthorizationCode</Operation>
```

The OAuth 2.0 operation executed by the policy.

Default:	If <Operation> is not specified, SAP API Management looks at the list of <SupportedGrantTypes>. Only operations on those grant types will be successful. In other words, you can omit <Operation> if you specify a <GrantType> in the <SupportedGrantTypes> list. If neither <Operation> nor <SupportedGrantTypes> are specified, the default grant type is authorization_code. That is, authorization_code grant type requests will succeed, but all others will fail.
Presence:	Optional
Type:	String
Valid values:	GenerateAccessToken - Generates an access token. GenerateAccessTokenImplicitGrant - Generates an access token for the implict grant type. GenerateAuthorizationCode - Generates an auth code. Used with the authorization code grant type. RefreshAccessToken - Exchange a new access token for a refresh token.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

	VerifyAccessToken - Verifies that an access token sent in a request is valid. InvalidateToken - Revokes an access token. After a token has been revoked, clients cannot use that token to call a protected API. ValidateToken - Reinstates or "approves" an access token that was previously revoked.

8.5.3.22 <PassWord> element

```
<PassWord>request.queryparam.password</PassWord>
```

In cases where the app user's password must be sent to the authorization server, this element lets you specify where SAP API Management should look for the password. For example, it could be sent as a query parameter, in an HTTP header, or a form parameter (default).

For example request.queryparam.password indicates that the password should be present as a query parameter, as, for example, ?password=changeit. To require the password in an HTTP header, for example, set this value to request.header.password.

Default:	request.formparam.password (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Any flow variable available to the the policy at runtime.
Used with grant types:	password

8.5.3.23 <RedirectUri> element

```
<RedirectUri>request.queryparam.redirect_uri</RedirectUri>
```

Specifies where SAP API Management should look for the redirect_uri parameter in the request.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

About redirection URIs

Redirection URIs are used with the authorization code and implicit grant types. The redirect URI tells the authorization server where to send an authorization code (for the auth code grant type) or access token (for the implicit grant type). It's important to understand when this parameter is required, when it is optional, and how it is used:

- (required) If a Callback URL is registered with the developer app that is associated with the request's client keys, and if the request_uri is present in the request, then the two must match exactly. If they do not match, an error is returned.
- (optional) If a Callback URL is registered, and the request_uri is missing from the request, then SAP API Management redirects to the registered Callback URL.
- (required) If a Callback URL is not registered, then the request_uri is required. Note that in this case, SAP API Management will accept ANY URL. This case can present a security issue, and therefore should only be used with trusted client apps. If client apps are not trusted, then the best practice is to always require the registration of a Callback URL.

You can send this parameter in a query parameter or in a header. The variable request.queryparam.redirect_uri indicates that the RedirectUri should be present as a query parameter, as, for example, ?redirect_uri=login.myapp.com. To require the RedirectUri in an HTTP header, for example, set this value to request.header.redirect_uri.

Default:	request.formparam.redirect_uri (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Any flow variable accessible in the policy at runtime
Used with grant types:	authorization_code implicit Also used with the GenerateAuthorizationCode operation.

8.5.3.24 <RefreshToken> element

```
<RefreshToken>request.queryparam.refreshtoken</RefreshToken>
```

When requesting an access token using a refresh token, you must supply the refresh token in the request. This element lets you specify where SAP API Management should look for the refresh token. For example, it could be sent as a query parameter, HTTP header, or form parameter (the default).

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The variable request.queryparam.refreshtoken indicates that the refresh token should be present as a query parameter, as, for example, ?refresh_token=login.myapp.com. To require the RedirectUri in an HTTP header, for example, set this value to request.header.refresh_token.

Default:	request.formparam.refresh_token (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Any flow variable accessible in the policy at runtime
Used with grant types:	refresh_token

8.5.3.25 <RefreshTokenExpiresIn> element

```
<RefreshTokenExpiresIn>1000</RefreshTokenExpiresIn>
```

Enforces the expiry time of refresh tokens in milliseconds. The expiry time value is a system generated value plus the <RefreshTokenExpiresIn>value. If <RefreshTokenExpiresIn> is set to -1, the refresh token is given an infinite lifetime. If <RefreshTokenExpiresIn> is not specified, the system applies a default value configured at the system level.

The expiry time can also be set at runtime using a reference to a flow variable. The flow variable can be retrieved from a header, query parameter, or form parameter (default). Or, it can be a hard-coded value.

For example request.queryparam.expires_in indicates that the expiry value should be present as a query parameter, as, for example, ?expires_in=360000. To require the value to come from an HTTP header, for example, set this value to request.header.expires_in.

The following stanza specifies a flow variable and a default value as well. Note that the flow variable value takes precedence over the specified default value.

```
<RefreshTokenExpiresIn ref="flow.variable">
    {default_value}
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
</RefreshTokenExpiresIn>
```

Default:	If not specified, the system applies a default value configured at the system level.
Presence:	Optional
Type:	Integer
Valid values:	Any integer, including -1 (which indicates an infinite expiry time).
Used with grant types:	authorization_code password refresh_token

8.5.3.26 <ResponseType> element

```
<ResponseType>request.queryparam.response_type</ResponseType>
```

This element informs SAP API Management which grant type the client app is requesting. It is used only with the authorization code and implicit grant type flows.

By default, SAP API Management looks for the response type value in a response_type query parameter. If you wish to override this default behavior, use the <ResponseType> element to configure a flow variable containing the response type value. For example, if you set this element to request.header.response_type, SAP API Management looks for the response type to be passed in the request header.

Default:	request.formparam.response_type (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional. Use this element if you wish to override the default behavior.
Type:	String
Valid values:	Either code (for the authorization code grant type) or token (for the implicit grant type)

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Used with grant types:	implicit Also used with the GenerateAuthorizationCode operation.

8.5.3.27 <ReuseRefreshToken> element

```
<ReuseRefreshToken>true</ReuseRefreshToken>
```

When set to true, the existing refresh token is reused until it expires. If false, a new refresh token is issued by SAP API Management when a valid refresh token is presented.

Default:	false
Presence:	optional
Type:	boolean
Valid values:	true or false
Used with grant type:	refresh_token

8.5.3.28 <Scope> element

```
<Scope>request.queryparam.scope</Scope>
```

If this element is present in an of the GenerateAccessToken or GenerateAuthorizationCode policies, it is used to specify which scopes to grant the token or code. These values are typically passed to the policy in the request from a client app. You can configure the element to take a flow variable, giving you the option to choose how the scopes are passed in a request. In the following example, request.queryparam.scope indicates that the scope should be present as a query parameter, as, for example, ?scope=READ. To require the scope in an HTTP header, for example, set this value to request.header.scope.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

If this element appears in a "VerifyAccessToken" policy, then it is used to specify which scopes the policy should enforce. In this type of policy, the value must be a "hard coded" scope name -- you can't use variables. For example:

<Scope>A B</Scope>

Default:	No scope
Presence:	Optional
Type:	String
Valid values:	If used with Generate* policies, a flow variable. If used with VerifyAccessToken, a space-separated list of scope names (strings).
Used with grant types:	authorization_code implicit password client_credentials Can also be used with the GenerateAuthorizationCode and VerifyAccessToken operations.

8.5.3.29 <State> element

```
<State>request.queryparam.state</State>
```

In cases where the client app must send the state information to the authorization server, this element lets you specify where SAP API Management should look for the state values. For example, it could be sent as a query parameter or in an HTTP header. The state value is typically used as a security measure to prevent CSRF attacks.

For example request.queryparam.state indicates that the state should be present as a query parameter, as, for example, ?state=HjoiuKJH32. To require the state in an HTTP header, for example, set this value to request.header.state.

Default:	No state
Presence:	Optional
Type:	String

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Valid values:	Any flow variable accessible to the policy at runtime
Used with grant types:	All Can also be used with the GenerateAuthorizationCode operation

8.5.3.30 <StoreToken> element

```
<StoreToken>true</StoreToken>
```

Set this element to true when the <ExternalAuthorization> element is true. The <StoreToken> element tells SAP API Management to store the external access token. Otherwise, it will not be persisted.

Default:	false
Presence:	Optional
Type:	Boolean
Valid values:	true or false
Used with grant types:	authorization_code password client_credentials

8.5.3.31 <SupportedGrantTypes> / <GrantType> element

```
<SupportedGrantTypes>
    <GrantType>authorization_code</GrantType>
    <GrantType>client_credentials</GrantType>
    <GrantType>implicit</GrantType>
    <GrantType>password</GrantType>
</SupportedGrantTypes>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Specifies the grant types supported by an OAuth token endpoint on SAP API Management. An endpoint may support multiple grant types (that is, a single endpoint can be configured to distribute access tokens for multiple grant types.) The grant type is passed in token requests in a grant_type parameter.

If no supported grant types are specified, then the only allowed grant types are authorization_code and implicit. See also the <GrantType>element (which is a higher-level element that is used to specify where SAP API Management should look for the grant_type parameter that is passed in a client request. SAP API Management will make sure that the value of the grant_type parameter matches one of the supported grant types).

Default:	authorization _code and implicit
Presence:	Required
Type:	String
Valid values:	client_credentials authorization_code password implicit

8.5.3.32 <Tokens> / <Token> element

Used with the ValidateToken and InvalidateToken operations.

The <Token> element identifies the flow variable that defines the source of the token to be revoked. If developers are expected to submit access tokens as query parameters named access_token, for example, use request.queryparam.access_token.

8.5.3.33 <UserName> element

```
<UserName>request.queryparam.user_name</UserName>
```

In cases where the app user name must be sent to the authorization server, this element lets you specify where SAP API Management should look for the end user name. For example, it could be sent as a query parameter or in an HTTP header.

For example request.queryparam.username indicates that the username should be present as a query parameter, as, for example, ?username=joe. To require the UserName in an HTTP header, for example, set this value to request.header.username.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Default:	request.formparam.password (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Any variable setting.
Used with grant types:	password

8.5.3.34 Verifying access tokens

Once a token endpoint is set up for an API proxy, a corresponding OAuthV2 policy that specifies the VerifyAccessToken operation is attached to the Flow that exposes the protected resource.

For example, to ensure that all requests to an API are authorized, the following policy enforces access token verification:

```
<OAuthV2 name="VerifyOAuthAccessToken">
  <Operation>VerifyAccessToken</Operation>
</OAuthV2>
```

The policy is attached to the API resource to be protected. To ensure that all requests to an API are verified, attach the policy to the ProxyEndpoint request PreFlow, as follows:

```
<PreFlow>
  <Request>
    <Step><Name>VerifyOAuthAccessToken</Name></Step>
  </Request>
</PreFlow>
```

The following optional elements can be used to override the default settings for the VerifyAccessToken operation.

Name	Description
Scope	A space-delimited list of scopes. Verification will succeed if at least one of the scopes listed is present in the access token. For example, the following policy will check the access token to

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description
	ensure that it contains at least one of the scopes listed. If READ or WRITE is present, verification will succeed. <OAuthV2 name="ValidateOauthScopePolicy"> <Operation>VerifyAccessToken</Operation> <Scope>READ WRITE</Scope> </OAuthV2>
AccessToken	The variable where the access token is expected to be located. For examplerequest.queryparam.accesstoken. By default, the access token is expected to be presented by the app in the Authorization HTTP header, according to the OAuth 2.0 specification. Use this setting if the access token is expected to be presented in a non-standard location, such as a query parameter, or an HTTP header with a name other than Authorization.

8.5.3.35 Specifying request variable locations

For each grant type, the policy makes assumptions about the location or required information in request messages. These assumptions are based on the the OAuth 2.0 specification. If your apps need to deviate from the OAuth 2.0 specification, then you can specify the expected locations for each parameter. For example, when handling an authorization code, you can specify the location of the authorization code, the client ID, the redirect URI, and the scope. These can be specified as HTTP headers, query parameters, or form parameters.

The example below demonstrates how you can specify the location of required authorization code parameters as HTTP headers:

```
...
<GrantType>request.header.grant_type</GrantType>
<Code>request.header.code</Code>
<ClientId>request.header.client_id</ClientId>
<RedirectUri>request.header.redirect_uri</RedirectUri>
<Scope>request.header.scope</Scope>
...
```

Or, if necessary to support your client app base, you can mix and match headers and query parameters:

```
...
<GrantType>request.header.grant_type</GrantType>
<Code>request.header.code</Code>
<ClientId>request.queryparam.client_id</ClientId>
<RedirectUri>request.queryparam.redirect_uri</RedirectUri>
<Scope>request.queryparam.scope</Scope>
...
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Only one location can be configured per parameter.

8.5.3.36 Flow variables

The flow variables defined in this table are populated when the respective OAuth policies are executed, and hence are available to other policies or applications executing in the API proxy flow.

8.5.3.37 VerifyAccessToken operation

These variables are set when the VerifyAccessToken policy operation executes.

API product variables will be populated automatically if the API products are configured with valid environment, proxies, and API resources (derived from the proxy.pathsuffix). Explicitly setting flow.resource.name variable is not required.

Where the API products are not configured with valid environments and API proxies, then flow.resource.name must explicitly be set to populate API product variables in the flow.

Variables	Description
organization_name	The name of the organization where the proxy is executing.
developer.id	The ID of the developer associated with the registered client app.
developer.app.name	The name of the developer associated with the registered client app.
client_id	The client ID of the registered client app.
grant_type	The grant type associated with the request.
token_type	The token type associated with the request.
access_token	The access token that is being verified.
accesstoken.{custom_attribute}	A named custom attribute in the access token.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

issued_at	The date the access token was issued.
expires_in	The expiration time for the access token.
status	The status of the access token (e.g., approved or revoked).
scope	The scope (if any) associated with the access token.
apiproduct.<custom_attribute_name>	A named custom attribute of the API product associated with the registered client app.
apiproduct.name	The name of the API product associated with the registered client app.
App-specific variables	
app.name	
app.id	
app.accessType	
app.callbackUrl	
app.status	
app.scopes	
app.appFamily	
app.apiproducts	
app.appParentStatus	
app.appType	

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

app.appParentId	
app.created_by	
app.created_at	
app.last_modified_at	
app.last_modified_by	
app.{custom_attributes}	
Developer-specific variables	Note : If the app.appType is "Company" ,then company attributes are populated and if app.appType is "Developer", then developer attributes are populated.
developer.id	
developer.userName	
developer.firstName	
developer.lastName	
developer.email	
developer.status	
developer.apps	
developer.created_by	
developer.created_at	
developer.last_modified_at	

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

developer.last_modified_by	
developer.{custom_attributes}	
Company-specific variables	Note : If the app.appType is "Company" ,then company attributes are populated and if app.appType is "Developer", then developer attributes are populated.
company.id	
company.displayName	
company.apps	
company.appOwnerStatus	
company.created_by	
company.created_at	
company.last_modified_at	
company.last_modified_by	
company.{custom_attributes}	

8.5.3.38 GenerateAuthorizationCode operation

These variables are set when the GenerateAuthorizationCode policy operation executes successfully:

Variable	Description
oauthv2authcode.{policy_name}.code	The authorization code generated when the policy executes.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Variable	Description
oauthv2authcode.{policy_name}.redirect_uri	The redirect URI associated with the registered client app.
oauthv2authcode.{policy_name}.scope	The optional OAuth scope passed in the client request.
oauthv2authcode.{policy_name}.client_id	The client ID passed in the client request.

8.5.3.39 GenerateAccessToken

These variables are set when the GenerateAccessToken policy operation executes successfully for the authorization code, password, and client credentials grant type flows:

Variable	Description
oauthv2accesstoken.{policy_name}.access_token	The access token generated when the policy executes.
oauthv2accesstoken.{policy_name}.token_type	Will be set to accesstoken.
oauthv2accesstoken.{policy_name}.expires_in	The expiry value for the token. See the <ExpiresIn> element for details.
oauthv2accesstoken.{policy_name}.refresh_token	The refresh token generated when the policy executes.
oauthv2accesstoken.{policy_name}.refresh_token_expires_in	The lifespan of the refresh token, in seconds.
oauthv2accesstoken.{policy_name}.refresh_token_issued_at	This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.
oauthv2accesstoken.{policy_name}.refresh_token_status	Set to approved or revoked.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.3.40 GenerateAccessTokenImplicitGrant

These variables are set when the GenerateAccessTokenImplicit policy operation executes successfully for the implicit grant type flow:

Variable	Description
oauthv2accesstoken.{policy_name}.access_token	The access token generated when the policy executes.
oauthv2accesstoken.{policy_name}.expires_in	The expiry value for the token. See the <ExpiresIn> element for details.

8.5.3.41 RefreshAccessToken operation

These variables are set when the RefreshAccessToken policy operation executes successfully:

Variable	Description
oauthv2accesstoken.{policy_name}.access_token	The access token generated when the policy executes.
oauthv2accesstoken.{policy_name}.token_type	Will be set to accesstoken.
oauthv2accesstoken.{policy_name}.expires_in	The expiry value for the token. See the <ExpiresIn> element for details.
oauthv2accesstoken.{policy_name}.refresh_token	The refresh token generated when the policy executes.
oauthv2accesstoken.{policy_name}.refresh_token_expires_in	The lifespan of the refresh token, in seconds.
oauthv2accesstoken.{policy_name}.refresh_token_issued_at	This time value is the string representation of the corresponding 32-bit timestamp quantity. For example, 'Wed, 21 Aug 2013 19:16:47 UTC' corresponds to the timestamp value of 1377112607413.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Variable	Description
oauthv2accesstoken.{policy_name}.refresh_token_status	Set to approved or revoked.

8.5.3.42 GenerateErrorResponse

These variables are set when the GenerateErrorResponse element is true.

Variable	Description
oauthV2.{policy-name}.failed	Set to true if the policy failed.
oauthv2.{policy_name}.{fault_name}	The name of the fault. For example, invalid_request
oauthv2.{policy_name}.{fault_cause}	The fault reason. For example: Token Expired

8.5.3.43 Error codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

The OAuthV2 Policy type defines the following error codes.

Error Code	Message
FailedToResolveOAuthConfig	Failed to resolve oauth config {0}
OperationRequired	Operation required
InvalidOperation	Operation {0} is invalid

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Error Code	Message
InsufficientScope	Required scope(s) : {0}
FailedToResolveClientId	Failed to resolve client id variable {0}
FailedToResolveAccessToken	Failed to resolve access token variable {0}
FailedToResolveRefreshToken	Failed to resolve refresh token variable {0}
FailedToResolveToken	Failed to resolve token variable {0}
FailedToResolveAuthorizationCode	Failed to resolve authorization code variable {0}
ExpiresInNotApplicableForOperation	ExpiresIn element is not valid for operation {0}
RefreshTokenExpiresInNotApplicableForOperation	RefreshToken ExpiresIn element is not valid for operation {0}
InvalidValueForExpiresIn	Invalid value for ExpiresIn element for operation {0}
InvalidValueForRefreshTokenExpiresIn	Invalid value for Refresh Token ExpiresIn element for operation {0}
InvalidGrantType	Invalid grant type: {0}
GrantTypesNotApplicableForOperation	Grant types are not applicable for operation {0}
UnSupportedGrantType	Unsupported Grant Type: {0}
InvalidParameter	Invalid required paramater: {0}
MissingParameter	Missing required paramater: {0}
SpecifyValueOrRefApiKey	Specify Api Key as value or ref in stepDefinition {0}
SpecifyAPIProduct	Specify the API Product to get details {0}

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Error Code	Message
FailedToFetchApiProduct	Failed to fetch api product for key {0}
FailedToResolveAPIKey	Failed to resolve API Key variable {0}
FailedToResolveAPIProduct	Failed to resolve API product variable {0}
InvalidTokenType	Valid token types : {0}, Invalid token type {1} in stepDefinition {2}
TokenValueRequired	Token value is required in stepDefinition {0}
FailedToResolveRealm	Failed to resolve realm {0}

8.5.4 Retrieve Token Attributes Using GetOAuthV2Info

SAP API Management generates and manages a set of OAuth resources for apps. Depending on the OAuth configuration for an organization, SAP API Management will generate and manage access tokens, authorization codes, and refresh tokens. For each OAuth resource that it generates, SAP API Management also creates and stores a profile.

The GetOauthV2Info policy type enables you to get attributes of tokens and to make them available to policies and code executing in an API proxy. This policy type can be useful when you need to configure dynamic, conditional behavior based on a value in an access token.

An access token has the following JSON representation on SP API Management:

```
{
  "issued_at" : "1372170159093",
  "application_name" : "ccd1803b-b557-4520-bd62-ddd3abf8e501",
  "scope" : "READ",
  "status" : "approved",
  "api_product_list" : "[FreeProduct]",
  "expires_in" : "3599",
  "developer.email" : "joe@weathersample.com",
  "organization_id" : "0",
  "refresh_token" : "82XMXgDyHTpFyXOaApj8C2AGIPnN2IZe",
  "client_id" : "deAVedE0W9Z9U35PAMaAJYphBJCGdrND",
  "access_token" : "shTUmeI1geSKin0TODcGLXBNe9vp",
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  "organization_name" : "apifactory",
  "refresh_count" : "0"
}
```

The properties of an access token profile are set as variables whenever a token is generated or validated. Sometimes, however, you will need to access these properties when no token generation or validation occurs. To do so, you can explicitly populate the access token profile by using the GetOAuthV2Info policy.

The AccessToken element value is set to the name of the variable where the access token can be located: either in the request message or in some other variable where it might be populated by am ExtractVariables policy.

8.5.4.1 Samples

You can provide a reference to a variable that contains the token. The policy configuration below will obtain the access token by reference to query parameter called access_token. The policy expects the access token to be presented by the app as a query parameter named access_token. The policy will use that access token to retrieve the associated profile from SAP API Management's token store. The access token's profile will then be used to populate a set of variables.

```
<GetOAuthV2Info name="GetTokenAttributes">
  <AccessToken ref="request.queryparam.access_token"></AccessToken>
</GetOAuthV2Info>
```

The oauthv2accesstoken variables are then populated with values specific to the access token.

The values of the variables could then be accessed, for example, by JavaScript. For example, to retrieve the scope(s) associated with an access token using JavaScript:

```
var scope = context.getVariable('oauthv2accesstoken.GetTokenAttributes.scope');
```

You can also retrieve attributes of an access token by using a policy of type GetOAuthV2Info and referring to a variable set by the execution of any OAuthV2 policy.

In some rare cases you may need to get the profile a statically configured token. You can do by providing the value of the access token as an element.

```
<GetOAuthV2Info name="GetTokenAttributes">
  <AccessToken>shTUmeI1geSKin0TODcGLXBNe9vp</AccessToken>
</GetOAuthV2Info>
```

You can do this with all other token types (client ID, authorization code, and refresh tokens) as well.

Retrieving Authorization Code Attributes

As with access tokens, you can retrieve authorization code attributes by using the AuthorizationCode element in a policy of type GetOAuthV2Info

```
<GetOAuthV2Info name="GetAuthCodeAttributes">
    <AuthorizationCode ref="{variable name}"/>
</GetOAuthV2Info>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Example

```
<GetOAuthV2Info name="GetAuthCodeAttributes">
    <AuthorizationCode ref="request.queryparam.code"></AuthorizationCode>
</GetOAuthV2Info>
```

Retrieving Refresh Token Attributes

```
<GetOAuthV2Info name="GetTokenAttributes">
    <RefreshToken ref="{variable name}"/>
</GetOAuthV2Info>
```

Example

```
<GetOAuthV2Info name="GetTokenAttributes">
 <RefreshToken ref="request.queryparam.refresh_token"></RefreshToken>
</GetOAuthV2Info>
```

Retrieving Static Attributes

```
<GetOAuthV2Info name="GetTokenAttributes">
 <AccessToken>shTUmeI1geSKin0TODcGLXBNe9vp</AccessToken>
</GetOAuthV2Info>
```

8.5.4.2 Element Reference

The element reference describes the elements and attributes of the GetOAuthV2Info policy.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetOAuthV2Info async="false" continueOnError="false" enabled="true" name="GetOAuthV2Info-1"
   <DisplayName>Get OAuth v2.0 Info 1</DisplayName>
   <AccessToken ref={some-variable}></AccessToken>
</GetOAuthV2Info
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.4.2.1 <GetOAuthV2Info> attributes

<GetOAuthV2Info async="false" continueOnError="false" enabled="true" name="Get-OAuth-v20-Info-1">

The following attributes are common to all policy parent elements.

Attribute	Description	Default	Presence
name	The internal name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the <DisplayName> element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Required
continueOnError	Set to false to return an error when a policy fails. This is expected behavior for most policies. Set to true to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to true to enforce the policy. Set to false to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to true, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to true improves API proxy performance. However, overusing async can hurt performance with too much thread switching.	false	Optional

8.5.4.2.2 <DisplayName> element

Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

<DisplayName>Policy Display Name</DisplayName>

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Default:	N/A If you omit this element, the the value of the policy'sname attribute is used.
Presence:	Optional
Type:	String

8.5.4.2.3 <AccessToken> element

Retrieves the profile for an access token. You pass in a either a variable that contains the access token string or a literal token string (rare case). In this example, the access token is retrieved from a query parameter passed in a request.

```
<AccessToken ref="request.queryparam.access_token"></AccessToken>
```

Default:	request.formparam.access_token (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Either a flow variable containing an access token string, or a literal string.

8.5.4.2.4 <AuthorizationCode> element

Retrieves the profile for an authorization code. You pass in a either a variable that contains the auth code string or a literal token string (rare case). In this example, the auth code is retrieved from a query parameter passed in a request. For a list of variables populated by this operation, see "Flow variables".

```
<AuthorizationCode ref="request.queryparam.authorization_code"></AuthorizationCode>
```

Default:	request.formparam.access_token (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Type:	String
Valid values:	Either a flow variable containing an auth code string, or a literal string.

8.5.4.2.5 <ClientId> element

Retrieves information related to a client ID. In this example, the client ID is retrieved from a query parameter passed in a request. For a list of variables populated by this operation, see "Flow variables".

```
<ClientId>ref="request.queryparam.client_id"></ClientId>
```

Default:	request.formparam.access_token (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Either a flow variable containing an auth code string, or a literal string.

8.5.4.2.6 <RefreshToken> element

Retrieves the profile for a refresh token. You pass in a either a variable that contains the refresh token string or a literal token string (rare case). In this example, the refresh token is retrieved from a query parameter passed in a request. For a list of variables populated by this operation, see "Flow variables".

```
<RefreshToken ref="request.queryparam.refresh_token"></RefreshToken>
```

Default:	request.formparam.access_token (a x-www-form-urlencoded and specified in the request body)
Presence:	Optional
Type:	String
Valid values:	Either a flow variable containing an refresh token string, or a literal string.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.4.3 Flow Variables

When an access token is granted or validated by a policy, the following attributes of the access token are set as variables. These variables are then available to other policies or code executing in the same Flow. For example, you might need to access these variables in another policy to enable custom behavior based on the scope associated with the access token.

Client ID Variables

These variables are populated when the <ClientId> operation executes:

oauthv2client.{policy_name}.client_id

oauthv2client.{policy_name}.client_secret

oauthv2client.{policy_name}.redirection_uris (* Note the spelling here -- 'redirection_uris')

oauthv2client.{policy_name}.developer.email

oauthv2client.{policy_name}.developer.app.name

oauthv2client.{policy_name}.developer.id

oauthv2client.{policy_name}.{custom_attribute_name}

Access Token Variables

These variables are populated when the <AccessToken> operation executes:

oauthv2accesstoken.{policy_name}.access_token

oauthv2accesstoken.{policy_name}.scope

oauthv2accesstoken.{policy_name}.refresh_token

oauthv2accesstoken.{policy_name}.accesstoken.{custom_attribute_name}

oauthv2accesstoken.{policy_name}.developer.id

oauthv2accesstoken.{policy_name}.developer.app.name

oauthv2accesstoken.{policy_name}.expires_in

oauthv2accesstoken.{policy_name}.status

Authorization Code Variables

These variables are populated when the <AuthorizationCode> operation executes:

oauthv2authcode.{policy_name}.client_id

oauthv2authcode.{policy_name}.organization_id

oauthv2authcode.{policy_name}.issued_at

oauthv2authcode.{policy_name}.expires_in

oauthv2authcode.{policy_name}.redirect_uri

oauthv2authcode.{policy_name}.status

oauthv2authcode.{policy_name}.state

oauthv2authcode.{policy_name}.scope

oauthv2authcode.{policy_name}.id

oauthv2authcode.{policy_name}.{custom_attribute_name}

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Refresh Token Variables

These variables are populated when the <RefreshToken> operation executes:

oauthv2authcode.{policy_name}.access_token

oauthv2authcode.{policy_name}.refresh_token

oauthv2authcode.{policy_name}.client_id

oauthv2authcode.{policy_name}.refresh_count

oauthv2authcode.{policy_name}.organization_name

oauthv2authcode.{policy_name}.refresh_token_expires_in

oauthv2authcode.{policy_name}.refresh_token_issued_at

oauthv2authcode.{policy_name}.refresh_token_status

oauthv2authcode.{policy_name}.developer.email

oauthv2authcode.{policy_name}.developer.id

oauthv2authcode.{policy_name}.developer.app.name

oauthv2authcode.{policy_name}.developer.app.id

oauthv2authcode.{policy_name}.{custom_attribute_name}

To support backward compatibility, the above flow variables prefixed with oauthv2accesstoken. `{policy_name}` are also available.

8.5.5 Set OAuth Token Attributes Using SetOAuthV2Info

SAP API Management generates and distributes OAuth access tokens to apps. SAP API Management stores those access tokens and uses them to authorize consumer apps. Some other types of OAuth tokens are also generated by SAP API Management. These include refresh tokens and authorization codes.

When SAP API Management generates these OAuth artifacts, it also generates 'profile' that contains metadata related to the token or code. For example, the default access token profile contains name/value pairs that define expiration time, the associated app and developer, and so on.

The JSON representation of an SAP API Management access token looks like the following:

```
{
  "issued_at" : "1372170159093",
  "application_name" : "ccd1803b-b557-4520-bd62-ddd3abf8e501",
  "scope" : "READ",
  "status" : "approved",
  "api_product_list" : "[FreeProduct]",
  "expires_in" : "3599",
  "developer.email" : "joe@weathersample.com",
  "organization_id" : "0",
  "refresh_token" : "82XMXgDyHTpFyXOaApj8C2AGIPnN2IZe",
  "client_id" : "deAVedE0W9Z9U35PAMaAJYphBJCGdrND",
  "access_token" : "shTUmeI1geSKin0TODcGLXBNe9vp",
  "organization_name" : "apifactory",
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  "refresh_count" : "0"
}
```

In some situations, you will need to update the profile of an access token. For example, you may want to embed a tag that is unique to you business. You might need to embed a department name, a customer ID or more technically, a session identifier, in the access token.

There are two ways to do this: Using an API call or using the SetOAuthV2Info policy. You can call the management API to directly update the access token's profile.

Use the policy when you need tokens to be updated at runtime, such as at the time when the token or code is generated by SAP API Management.

8.5.5.1 Samples

Below is an example policy used to update an OAuth 2.0 access token. The example below locates the access token on the request message by looking for a query parameter called access_token. When an access token is presented by a client app, the policy below will locate the access token in the query parameter. It will then update the access token's profile in two ways: it will added a property called department.id to the profile. It will also modify the access token's scope property to the value READ, WRITE.

```
<SetOAuthV2Info name="SetOAuthV2Info">
  <AccessToken ref="request.queryparam.access_token"></AccessToken>
  <Attributes>
    <Attribute name="department.id"
ref="request.queryparam.department_id"></Attribute>
    <Attribute name="scope" ref="">READ, WRITE</Attribute>
  </Attributes>
</SetOAuthV2Info>
```

> **Note**
>
> If an attribute already exists in the access token profile, then it will be updated with the new value in the policy. If an attribute does not exist, then the attribute will be added to the access token's profile.

8.5.5.2 Element Reference

The element reference describes the elements and attributes of the SetOAuthV2 policy.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SetOAuthV2Info async="false" continueOnError="false" enabled="true"
name="SetOAuthV2Info-1">
    <DisplayName>Set OAuth v2.0 Info 1</DisplayName>
    <AccessToken ref={some-variable}></AccessToken>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <Attributes/>
</SetOAuthV2Info
```

8.5.5.2.1 <SetOAuthV2Info> attributes

<SetOAuthV2Info async="false" continueOnError="false" enabled="true" name="Set-OAuth-v20-Info-1">

The following attributes are common to all policy parent elements.

Attribute	Description	Default	Presence
name	The internal name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the <DisplayName> element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Required
continueOnError	Set to false to return an error when a policy fails. This is expected behavior for most policies. Set to true to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to true to enforce the policy. Set to false to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to true, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to true improves API proxy performance. However, overusing async can hurt performance with too much thread switching.	false	Optional

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.5.2.2 <DisplayName> element

Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

<DisplayName>Policy Display Name</DisplayName>

Default: N/A
If you omit this element, the the value of the policy'sname attribute is used.

Presence: Optional

Type: String

8.5.5.2.3 <AccessToken> element

Identifies the variable where the access token is located. For example, if the access token is attached to request message as a query parameter, specify request.queryparam.access_token. You can use any valid variable that references the token. Or, could pass in the literal token string (rare case).

<AccessToken ref="request.queryparam.access_token"></AccessToken>

Default: N/A

Presence: Required

Type: String

8.5.5.2.4 Attributes

Attribute	Description
ref	An access token variable. Typically, retrieved from a flow variable.

8.5.5.2.5 <Attributes> element

A set of attributes in the access token profile that will be modified or augmented.

Default: N/A

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Presence: Required

Type: N/A

8.5.5.2.6 <Attributes>/<Attribute> element

An individual attribute to update.

The name attribute identifies the property of the access token profile to be updated. For example, to modify the access token's scope property, specify scope as the value of the name attribute.

The ref attribute specifies either variable or a static whose value will be used as the value of the access token profile property that will be updated. For example to update the attribute scope with the value READ, WRITE:

```
<Attributes>
  <Attribute name="department.id" ref="request.queryparam.department_id"></Attribute>
  <Attribute name="scope" ref="">READ, WRITE</Attribute>
</Attributes>
```

Default: N/A

Presence: Optional

Type: N/A

8.5.5.2.7 Attributes

Attribute	Description	Default	Presence
name	The name of the profile attribute to add or change.	N/A	
ref	The value to assign to the profile attribute.	N/A	Optional

8.5.5.3 Policy-specific Variables

On success, the following flow variables will be set:

- **`oauthv2accesstoken.{policyName}.access_token`**
- **`oauthv2accesstoken.{policyName}.client_id`**
- **`oauthv2accesstoken.{policyName}.refresh_count`**

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- `oauthv2accesstoken.{policyName}.organization_name`
- `oauthv2accesstoken.{policyName}.expires_in`
- `oauthv2accesstoken.{policyName}.refresh_token_expires_in`
- `oauthv2accesstoken.{policyName}.issued_at`
- `oauthv2accesstoken.{policyName}.status`
- `oauthv2accesstoken.{policyName}.api_product_list`
- `oauthv2accesstoken.{policyName}.token_type`
- `oauthv2accesstoken.{policyName}.{custom_attribute_name}`

On failure, following variable will be set:

- `oauthv2accesstoken.{policyName}.failed: true`

8.5.6 Delete Authorization Code or Access Token using DeleteOAuthV2Info

8.5.6.1 Samples

Below is an example policy used to delete an OAuth 2.0 access token. The example below locates the access token to delete on the request message by looking for a header called access_token.

```
<DeleteOAuthV2Info name="DeleteAccessToken">
     <AccessToken ref="request.header.access_token"></AccessToken>
</DeleteOAuthV2Info>
```

Below is an example policy used to delete an OAuth 2.0 authorization code. The example below locates the auth code to delete on the request message by looking for a query parameter called code.

```
<DeleteOAuthV2Info name="DeleteAuthCode">
     <AuthorizationCode ref="request.queryparam.code"></AuthorizationCode>
</DeleteOAuthV2Info>
```

8.5.6.2 Element Reference

The element reference describes the elements and attributes of the DeleteOAuthV2Info policy.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DeleteOAuthV2Info async="false" continueOnError="false" enabled="true"
name="DeleteOAuthV2Info-1">
    <DisplayName>Delete OAuth v2.0 Info 1</DisplayName>
    <AccessToken ref={some-variable}></AccessToken>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <!--<AuthorizationCode ref={some-variable}></AuthorizationCode>-->
    <Attributes/>
</DeleteOAuthV2Info
```

8.5.6.2.1 <DeleteOAuthV2Info> attributes

```
<DeleteOAuthV2Info async="false" continueOnError="false" enabled="true" name="Delete-OAuth-v20-Info-1">
```

Attribute	Description	Default	Presence
async	Set to true to specify that the policy should be run in a thread pool different than the pool servicing the request/response flow. Default is false. This setting is only used for for internal optimization.	false	Optional
continueOnError	Most policies are expected to return an error when a failure occurs. By setting this attribute to true, Flow execution continues on failure.	false	Optional
enabled	Determines whether a policy is enforced or not. If set to false, a policy is 'turned off', and not enforced (even though the policy remains attached to a Flow).	true	Optional
name	The internal name of the policy. This name is referenced in Step elements to attach the policy to a Flow. Note: Characters you can use in the name are restricted to: A-Z0-9._\-$ %. The Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.	N/A	Required

8.5.6.2.2 <AccessToken> element

Identifies the variable where the access token to delete is located. For example, if the access token is attached to request message as a query parameter called "access_token", specify request.queryparam.access_token. You can use any valid variable that references the token. Or, could pass in the literal token string (rare case).

<AccessToken ref="request.queryparam.access_token"></AccessToken>

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Default: N/A

Presence: Either <AccessToken> or <AuthorizationCode> is required.

Type: String

8.5.6.2.3 Attributes

Attribute	Description	Default	Presence
ref	An access token variable. Typically, retrieved from a flow variable. For example: request.header.token or request.queryparam.token.	N/A	Optional

8.5.6.2.4 <AuthorizationCode> element

Identifies the variable where the authorization code to delete is located. For example, if the auth code is attached to request message as a query parameter called "code", specify request.queryparam.code. You can use any valid variable that references the token. Or, could pass in the literal token string (rare case).

```
<AccessToken ref="request.queryparam.access_token"></AccessToken>
```

Default: N/A

Presence: Either <AccessToken> or <AuthorizationCode> is required.

Type: String

8.5.6.2.5 Attributes

Attribute	Description	Default	Presence
ref	An access token variable. Typically, retrieved from a flow variable. For example: request.header.code or request.queryparam.code.	N/A	Optional

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.6.2.6 <DisplayName> element

A natural-language name that labels the policy in the management UI proxy editor. If omitted, the policy name attribute is used.

<DisplayName>DeleteOAuthV2Info 1</DisplayName>

Default:	The value of the policy's name attribute.
Presence:	Optional
Type:	String

Output

On success, the policy returns a 200 status.

On failure, the policy returns 404 and output similar to the following (depending on whether you are deleting an access token or an auth code):

```
HTTP/1.1 404 Not Found
Content-Type: application/json
Content-Length: 144
Connection: keep-alive

{"fault":{"faultstring":"Invalid Authorization
Code","detail":{"errorcode":"keymanagement.service.invalid_request-
authorization_code_invalid"}}}
```

8.5.7 OAuth HTTP Status Codes

This topic provides HTTP status codes and their related reason phrases you may encounter when OAuth throws errors in SAP API Management.

For policy-specific error codes, see:

OAuthV2 policy

OAuth v1.0a policy

8.5.7.1 Authorization Code

8.5.7.1.1 Invalid Redirect URI

```
HTTP/1.1 400 Bad Request
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
{"ErrorCode" : "invalid_request", "Error" :"Invalid redirection uri
http://www.invalid_example.com"}
```

8.5.7.1.2 No Redirect URI

```
HTTP/1.1 400 Bad Request {"ErrorCode" : "invalid_request", "Error" :"Redirection URI
is required"}
```

8.5.7.1.3 Invalid Key

```
HTTP/1.1 401 Unauthorized {"ErrorCode" : "invalid_request", "Error" :"Invalid client
id : AVD7ztXReEYyjpLFkkPiZpLEjeF2aYAz. ClientId is Invalid"}
```

8.5.7.1.4 Missing Key

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"The request is missing a required
parameter : client_id"}
```

8.5.7.1.5 Invalid Response Type

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Response type must be code"}
```

8.5.7.1.6 Missing Response Type

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"The request is missing a required
parameter : response_type"}
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.7.2 Generate AccessToken

8.5.7.2.1 Invalid Auth Code

```
HTTP status: 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Invalid Authorization Code"}
```

8.5.7.2.2 No Redirect URI

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Required param : redirect_uri"}
```

8.5.7.2.3 Invalid Redirect URI

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Invalid redirect_uri : oob"}
```

8.5.7.2.4 Invalid Client ID

```
HTTP/1.1 401 Unauthorized
{"ErrorCode" : "invalid_client", "Error" :"Client identifier is required"}
```

8.5.7.2.5 No Client ID

```
HTTP/1.1 401 Unauthorized
{"ErrorCode" : "invalid_client", "Error" :"Client identifier is required"}
```

8.5.7.2.6 Invalid GrantType

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Unsupported grant type :
client_credentials_invalid"}
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.7.2.7 No Username

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Required param : username"}
```

8.5.7.2.8 No Password

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Required param : password"}
```

8.5.7.2.9 No GrantType (Custom Policy)

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Required param : grant_type"}
```

8.5.7.2.10 No AuthCode

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Required param : code"}
```

8.5.7.3 Implicit

8.5.7.3.1 Invalid Client ID

```
HTTP/1.1 401 Unauthorized
{"ErrorCode" : "invalid_request", "Error" :"Invalid client id :
AVD7ztXReEYyjpLFkkPiZpLEjeF2aYAz. ClientId is Invalid"}
```

8.5.7.3.2 No Client ID

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"The request is missing a required
parameter : client_id"}
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.7.3.3 Invalid Response Type

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Response type must be token"}
```

8.5.7.3.4 No Response Type

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"The request is missing a required
parameter : response_type"}
```

8.5.7.3.5 Invalid Redirect URI

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Invalid redirection uri
http://www.invalid_example.com"}
```

8.5.7.3.6 No Redirect URI

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Redirection URI is required"}
```

8.5.7.4 Refresh Token

8.5.7.4.1 Invalid RefreshToken

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Invalid Refresh Token"}
```

8.5.7.4.2 Invalid Scope

```
HTTP/1.1 400 Bad Request
{"ErrorCode" : "invalid_request", "Error" :"Invalid Scope"}
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.7.4.3 Invalid Client ID

```
HTTP/1.1 401 Unauthorized
{"ErrorCode" : "invalid_client", "Error" :"Client identifier is required"}
```

8.5.7.4.4 No Client ID

```
HTTP/1.1 401 Unauthorized
{"ErrorCode" : "invalid_client", "Error" :"Client identifier is required"}
```

8.5.7.5 Verify AccessToken

8.5.7.5.1 Invalid AccessToken

```
HTTP/1.1 401 Unauthorized
{"fault":{"faultstring":"Invalid Access
Token","detail":{"errorcode":"keymanagement.service.invalid_access_token"}}}
```

8.5.7.5.2 Invalid Resource

```
HTTP/1.1 401 Unauthorized
{"fault":{"faultstring":"APIResource \/facebook\/acer does not
exist","detail":{"errorcode":"keymanagement.service.apiresource_doesnot_exist"}}}
```

8.5.7.5.3 Invalid Scope

```
HTTP/1.1 403 Forbidden
{"fault":{"faultstring":"Required scope(s) :
VerifyAccessToken.scopeSet","detail":{"errorcode":"steps.oauth.v2.InsufficientScope"}}
}
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.7.5.4 No Auth Header

```
HTTP/1.1 401 Unauthorized
{"fault":{"faultstring":"Invalid access
token","detail":{"errorcode":"oauth.v2.InvalidAccessToken"}}}
```

8.5.7.5.5 No match for ApiProduct (With Env & Proxy Configured)

```
HTTP/1.1 401 Unauthorized
{"fault":{"faultstring":"Invalid API call as no apiproduct match
found","detail":{"errorcode":"keymanagement.service.InvalidAPICallAsNoApiProductMatchF
ound"}}}
```

8.5.7.5.6 Access token expired

```
HTTP/1.1 401 Unauthorized
{"fault":{"faultstring":"Access Token
expired","detail":{"errorcode":"keymanagement.service.access_token_expired"}}}
```

8.5.7.5.7 Access token revoked

```
HTTP/1.1 401 Unauthorized
{"fault":{"faultstring":"Access Token not
approved","detail":{"errorcode":"keymanagement.service.access_token_not_approved"}}}
```

8.5.8 Authenticate and Authorize Using SAML 2.0

The Security Assertion Markup Language (SAML) specification defines formats and protocols that enable applications to exchange XML-formatted information for authentication and authorization.

A "security assertion" is a trusted token that describes an attribute of an app, an app user, or some other participant in a transaction. Security assertions are managed and consumed by two types of entities:

- Identity providers: Generate security assertions on behalf of participants
- Service providers: Validate security assertions through trusted relationships with identity providers

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The API platform can act as an identity provider and as a service provider. It acts as an identity provider by generating assertions and attaching them to request messages, making those assertions available for processing by backend services. It acts as a service provider by validating assertions on inbound request messages.

8.5.8.1 Inbound authentication and authorization

The SAML policy type enables API proxies to validate SAML assertions that are attached to inbound SOAP requests. The SAML policy validates incoming messages that contain a digitally-signed SAML assertion, rejects them if they are invalid, and sets variables that allow additional policies, or the backend services itself, to further validate the information in the assertion.

Sample

The following is a sample policy of type ValidateSAMLAssertion.

```
<ValidateSAMLAssertion name="SAML">
  <TrustStore>TrustStoreName</TrustStore>
  <ValidateSigner>true</ValidateSigner>
  <RemoveAssertion>false</RemoveAssertion>
  <AssertionLocation>xpath</AssertionLocation>
  <Message>request</Message>
  <IgnoreContentType>false</IgnoreContentType>
</ValidateSAMLAssertion>
```

Policy processing:

1. The policy checks the inbound message to verify that the request's media type is XML, by checking if the content type matches the formats text/(.*+)?xml or application/(.*+)?xml. If the media type is not XML, or if "IgnoreContentType" is not set, then the policy will raise a fault.
2. The policy will parse the XML. If parsing fails then it will raise a fault.
3. The policy will validate the XML digital signature, using the values of TrustStore and ValidateSigner as described above. If validation fails then it will raise a fault.
4. If present, the policy will check the current timestamp against the NotBefore and NotOnOrAfter elements in the assertion, as described in SAML Core section 2.5.1.
5. Any additional rules for processing the "Conditions" as described in SAML Core section 2.5.1.1.

Once the policy has completed without raising a fault, the developer of the proxy can be sure of the following:

- The digital signature on the assertion is valid and was signed by a trusted CA
- The assertion is valid for the current time period
- The subject and issuer of the assertion will be extracted and set in flow variables. It is the responsibility of other policies to use these values for additional authentication, such as checking that the subject name is valid, or passing it to a target system for validation.
- Other policies, such as ExtractVariables, may be used to parse the raw XML of the assertion for more complex validation.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.8.2 Supported versions of SAML

The SAML policy type supports SAML assertions that match version 2.0 of the SAML Core Specification and Version 1.0 of the WS-Security SAML Token Profile specification.

8.5.8.3 Outbound token generation

The SAML policy type enables API proxies to attach SAML assertions to outbound XML requests. Those assertions are then available to enable backend services to apply further security processing for authentication and authorization.

Sample

```
<GenerateSAMLAssertion name="SAML">
  <AssertionLocation>xpath</AssertionLocation>
  <Message>request</Message>
  <IgnoreContentType>false</IgnoreContentType>
  <KeyStore>
    <Name>keystorename</Name>
    <Alias>alias</Alias>
  </KeyStore>
  <Subject ref="reference">subject name</Subject>
  <Issuer ref="reference">issuer name</Issuer>
  <Template>
    <!-- A lot of XML goes here, in CDATA, with {} around
         each variable -->
  </Template>
</GenerateSAMLAssertion>
```

Policy processing:

1. If the message is not XML, and IgnoreContentType is not set to true, then raise a fault.
2. If "Template" is set, then process the template as described for the AssignMessage policy. If any variables are missing and IgnoreUnresolvedVariables is not set, then raise a fault.
3. If "Template" is not set, then construct an assertion that includes the values of the Subject and Issuer parameters or their references.
4. Sign the assertion using the specified key.
5. Add the assertion to the message at the specified XPath.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.8.4 Configuring a ValidateSAMLAssertion Policy

Field Nam	Description
name	The name of the policy instance. The name must be unique in the organization. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.
TrustStore	The name of the TrustStore that contains trusted X.509 certificates used to validate digital signatures on SAML assertions.
ValidateSigner	A boolean that can be set to true or false. When true, the policy validates the digital signature on the SAML assertion against the digital certificates in the TrustStore named in the TrustStore element.
RemoveAssertion	A boolean that can be set to true or false. When true, the SAML assertion will be stripped from the request message before the message is forwarded to the backend service.
AssertionLocation	An XPath expression that indicates the element on the inbound XML document from which the policy can extract the SAML assertion.
Message	The target of the policy. Valid values are message, request, and response. When set to message, the policy conditionally retrieves the message object based on the attachment point of the policy. When attached to the request Flow, the policy resolvesmessage to request, and when attached to the response Flow, the policy resolvesmessage to response. **Note** The ValidateSAMLAssertion can only be attached to the ProxyEndpoint request Flow.
IgnoreContentType	A boolean that can be set to true or false. By default, the assertion will not be generated if the content type of the message is not an XML Content-type. If this is set totrue then the message will be treated as XML regardless of the Content-type.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.8.5 Configuring a GenerateSAMLAssertion Policy

Field Name	Description
AssertionLocation	An XPath expression that indicates the element on the outbound XML document to which the policy will attach the SAML assertion.
Message	The target of the policy. Valid values are message, request, and response. When set to message, the policy conditionally retrieves the message object based on the attachment point of the policy. When attached to the request Flow, the policy resolvesmessage to request, and when attached to the response Flow, the policy resolvesmessage to response. Note that GenerateSAMLAssertion can only be attached to the TargetEndpoint request Flow. SO, when using the GenerateSAMLAssertion policy, you should set this value to request.
IgnoreContentType	A boolean that can be set to true or false. By default, the assertion will not be generated if the nedia type of the message is not an XML Content-type. If this is set totrue then the message will be treated as XML regardless of the Content-type.
KeyStore	The name of the KeyStore that contains the private key and the alias of the private key used to digitally sign SAML assertions.
Subject	The unique identifier of the subject of the SAML assertion. If the optional ref attribute is present, then the value of Subject will be assigned at runtime based on the specified variable. If the optional ref attribute is present, then the value of Subject will be used.
Issuer	The unique identifier of the identity provider. If the optional ref attribute is present, then the value of Issuer will be assigned at runtime based on the specified variable. If the optional ref attribute is present, then the value of Issuer will be used.
Template	If present, then the assertion will be generated by running this template, replacing everything denoted {} with the corresponding variable, and then digitally signing the result. The template is processed following the AssignMessage policy rules. See Generate or modify messages using AssignMessage.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.8.6 Policy-specific Variables

There are many pieces of information that may be specified in a SAML assertion. The SAML assertion itself is XML that can be parsed using the ExtractMessage policy and other mechanisms in order to implement more complex validations.

See Extract message content using ExtractVariables.

Variable	Description
saml.id	The SAML assertion ID
saml.issuer	The "Issuer" of the assertion, converted from its native XML type to a string
saml.subject	The "Subject" of the assertion, converted from its native XML type to a string
saml.valid	Returns true or false based on the result of the validity check
saml.attribute.{attribute_name}	The value of the named "Attribute" present in the assertion, converted from its native XML to a string
saml.attributeNames	The names of all the "Attributes" present in the assertion, in a comma-separated list
saml.issueInstant	IssueInstant
saml.subjectFormat	Subject format
saml.scmethod	Subject confirmation method
saml.scdaddress	Subject confirmation data address
saml.scdinresponse	Subject confirmation data in response
saml.scdrcpt	Subject confirmation data recipient
saml.authnSnooa	AuthnStatement SessionNotOnOrAfter
saml.authnContextClassRef	AuthnStatement AuthnContextClassRef
saml.authnInstant	AuthnStatement AuthInstant
saml.authnSessionIndex	AuthnStatement Session Index

8.5.8.7 Policy-specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
}
```

8.5.9 Authenticate to Backend Services Using Basic Authentication

8.5.9.1 Encode and decode credentials using Basic Authentication

The Basic Authentication policy type enables you to use the lightweight Basic Authentication for last-mile security. The policy takes the username and password defined in the policy, Base64 encodes them, and adds the resulting value to a variable. The resulting value is in the form Basic `Base64EncodedString`. You typically write this value to an HTTP header, such as th `Authorization` header.

The policy also lets you decode credentials stored in a Base64 encoded string into a username and password.The username and password are commonly stored the key/value store and then read from the key/value store at runtime. For details on using key/value store, see Persist data using KeyValueMap.

The policy also lets you decode credentials on an inbound request.

> **Note**
>
> This policy does not enforce Basic Authentication on a request to an API proxy. Instead, you use it to Base64 encode/decode credentials, typically when connecting to a backend server or using a service callout policy, such as the Call services or APIs using ServiceCallout that requires Basic Authentication.

8.5.9.1.1 Samples

The following samples illustrate how you can use the Basic Authentication policy to encode and decode basic auth credentials.

Outbound encoding

In the configuration below, username and password are from the variables specified in the `ref` attributes on the `<User>` and `<Password>` elements. The variables must be set by another policy before this policy executes. Typically, the variables are populated by values that are read from a key/value map. (See Persist data using KeyValueMap.

```
<BasicAuthentication name="ApplyBasicAuthHeader">
  <DisplayName>ApplyBasicAuthHeader</DisplayName>
  <Operation>Encode</Operation>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <User ref="BasicAuth.credentials.username" />
  <Password ref="BasicAuth.credentials.password" />
  <AssignTo createNew="false">request.header.Authorization</AssignTo>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
</BasicAuthentication>
```

This configuration results in an `Authorization` HTTP header being applied to the outbound request message. (The outbound request is the message that the API proxy sends to the backend service specified in the TargetEndpoint configuration). The `<User>` and `<Password>` are concatenated with a colon prior to `Base64` encoding.

```
Authorization: Basic TXlVc2VybmFtZTpNeVBhc3N3b3Jk
```

Imagine that you have a key/value map with the following entry:

```
{
  "entry" : [ {
    "name" : "username",
    "value" : "MyUsername
  }, {
    "name" : "password",
    "value" : "MyPassword
  } ],
  "name" : "BasicAuthCredentials"
}
```

Then you would attach the following KeyValueMapOperations policies before the BasicAuthentication policy. This way, the values for `<User>` and `<Password>` are extracted from the key/value store and populated into the variables `BasicAuth.credentials.username and BasicAuth.credentials.password`.

```
<KeyValueMapOperations name="getUsername" mapIdentifier="BasicAuthCredentials">
<Scope>apiproxy</Scope>
  <Get assignTo="credentials.username" index='1'>
    <Key>
      <Parameter ref="BasicAuth.credentials.username"/>
    </Key>
  </Get>
</KeyValueMapOperations>
```

and

```
<KeyValueMapOperations name="getPassword" mapIdentifier="BasicAuthCredentials">
<Scope>apiproxy</Scope>
  <Get assignTo="credentials.password" index='1'>
    <Key>
      <Parameter ref="BasicAuth.credentials.password"/>
    </Key>
  </Get>
</KeyValueMapOperations>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Inbound decoding

In this policy sample, the policy decodes the username and password from the `Authorization` HTTP header, as specified by the <Source> element. The Base64 encoded string must be in the form Basic `Base64EncodedString`.

```
.<BasicAuthentication name="DecodeBaseAuthHeaders">
  <DisplayName>Decode Basic Authentication Header</DisplayName>
  <Operation>Decode</Operation>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <User ref="request.header.username" />
  <Password ref="request.header.password" />
  <Source>request.header.Authorization</Source>
</BasicAuthentication>
```

The policy writes the decoded username to the `request.header.username` variable and the decoded password to the `request.header.password` variable.

8.5.9.2 Configuring the BasicAuthentication Policy

This policy has two modes of operations:

- Encode: Base64 encodes a username and password stored in variables
- Decode: Decodes the username and password from a Base64 encoded string

The username and password are commonly stored the key/value store and then read from the key/value store at runtime. For details on using key/value store, see Persist data using KeyValueMap.

> **Note**
>
> Basic authentication does not provide confidentiality. Always send credentials over a TLS-enabled (HTTPS) connection to your backend service.

8.5.9.2.1 Element reference

The element reference describes the elements and attributes of the BasicAuthentication policy.

```
<BasicAuthentication async="false" continueOnError="false" enabled="true" name="Basic-Authentication-1">
   <DisplayName>Basic Authentication 1</DisplayName>
   <Operation>Encode</Operation>
   <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <User ref="request.queryparam.username" />
    <Password ref="request.queryparam.password" />
    <AssignTo createNew="false">request.header.Authorization</AssignTo>
    <Source>request.header.Authorization</Source>
</BasicAuthentication>
```

8.5.9.2.2 <BasicAuthentication> attributes

```
<BasicAuthentication async="false" continueOnError="false" enabled="true" name="Basic-Authentication-1">
```

Configure BasicAuthentication using the following elements.

Field Name	Description	Default	Required
name	The unique name of the policy. Use this name to reference the policy within a Step definition element. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the `<DisplayName>` element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Yes
continueOnError	Set to `false` to return an error when a policy fails. This is expected behavior for most policies. Set to `true` to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to `true` to enforce the policy. Set to `false` to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to `true`, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and	false	Optional

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name	Description	Default	Required
	finishes handling the message flow. In some cases, setting async to `true` improves API proxy performance. However, overusing async can hurt performance with too much thread switching. To use asynchronous behavior in API proxies, see Java Callout Policy.		

8.5.9.2.3 <DisplayName> element

Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

```
<DisplayName>Policy Display Name</DisplayName>
```

Default: N/A
If you omit this element, the the value of the policy's `name` attribute is used.

Presence: Optional

Type: String

8.5.9.2.4 <Operation> element

Determines whether the policy Base64 encodes or decodes credentials.

```
<Operation>Encode</Operation>
```

Default: N/A

Presence: Required

Type: String.
Valid values include:

- Encode
- Decode

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.9.2.5 <IgnoreUnresolvedVariables> element

When set to `true`, the policy will not throw an error if a variable cannot be resolved. When used in the context of a BasicAuthentication policy, this setting is usually set to `false` because it is generally beneficial to throw an error if a username or password cannot be found in the variables specified.

```
<IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
```

Default: N/A

Presence: Required

Type: Boolean

8.5.9.2.6 <User> element

- For encoding, use the `<User>` element to specify the variable containing the username. Username and password values are concatenated with a colon prior to Base64 encoding.
- For decoding, specify the variable where the decoded username is written.

```
<User ref="request.queryparam.username" />
```

Default: N/A

Presence: Required

Type: N/A

8.5.9.2.7 Attributes

Attribute	Description	Default	Presence
ref	The variable from which the policy dynamically reads the username (encode) or writes the username (decode).	N/A	Required

8.5.9.2.8 <Password> element

For encoding, use the <Password> element to specify the variable containing the password.

For decoding, specify the variable where the decoded password is written.

```
<Password ref="request.queryparam.password" />
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Default: N/A

Presence: Required

Type: N/A

8.5.9.2.9 Attributes

Attribute	Description	Default	Presence
ref	The variable from which the policy dynamically reads the password (encode) or writes the password (decode).	N/A	Required

8.5.9.2.10 <AssignTo> element

For encoding, this is the variable where the encoded value, in the form Basic `Base64EncodedString` is written. For example, `request.header.Authorization`, corresponding to the `Authorization` header.

```
<AssignTo createNew="false">request.header.Authorization</AssignTo>
```

Default: N/A

Presence: Required

Type: N/A

8.5.9.2.11 Attributes

Attribute	Description	Default	Presence
createNew	For the BasicAuthentication policy, you will usually set this attribute to `false`. In the context of BasicAuthentication, you do not want to require that the policy generate a new message. Instead, you want the destination variable to be added to the existing request message.	False	Optional

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.9.2.12 <Source> element

For decoding, the variable containing the Base64 encoded string, in the form Basic `Base64EncodedString`. For example, specify `request.header.Authorization`, corresponding to the `Authorization` header.

```
<Source>request.header.Authorization</Source>
```

Default: N/A

Presence: Required

Type: N/A

8.5.9.2.13 Flow variables

The following flow variable is set when the policy fails:

- `BasicAuthentication.{policy_name}.failed (with a value of true)`

> **Note**
>
> If you see variables called `BasicAuthentication.{policy_name}.fault.cause` and `BasicAuthentication.{policy_name}.fault.name`, they are not currently being used by SAP API Management.

8.5.9.2.14 Error codes

The default format for error codes returned by policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

This policy defines the following error codes. For guidance on handling errors, see Fault Handling.

Error Code	Message / Description
`InvalidBasicAuthenticationSource`	`Source variable : request.header.Authorization for basic authentication decode policy is not valid` Occurs on a decode when the variable containing the Base64 encoded string does not contain a valid value.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Error Code	Message / Description
`UnresolvedVariable`	`Unresolved variable : request.queryparam.username` Occurs when IgnoreUnresolvedVariables is false for the policy and the source variables for the decode or encode are not present.

8.5.10 Manage Client Access Using IP-based Access Control

Sometimes you will find it necessary to limit access to your APIs from a specific network or computer. You might also need to identify specific IP addresses that cannot access your API, such as a network address that has been identified as a malicious actor.

Configuring access control based on IP addresses is a useful way to provide coarse-grained control. For example, if you only want computers under the control of your enterprise to access the APIs exposed in your test environment, you can allow (or whitelist) the IP address range for your internal network. Developers working from home can access these APIs using VPN.

> **Note**
>
> If you block a specific address, the computer using that IP address can still directs requests to your API through a proxy. It is also possible to spoof (falsify) the IP address of a request, so the request appears to come from a reputable source.

SAP API Management enables you to attach an Access Control policy to your APIs (request path), where you can define a range of IPs and allow IPs within the specified range to access an operation or service.

The configuration and execution of an Access Control policy involves the following tasks:

- Define a set of match rules with one of two actions (ALLOW or DENY) associated with each.
- For each match rule, specify the IP address (SourceAddress element).
  - Configure a mask for each IP address. You allow or deny access based on a mask value on the IP address. For more information about IP addresses, see IP addresses and dotted notation.
  - If the X-FORWARDED-FOR header is present in the request address, then the header value is considered the source address.
- Specify the order in which the rules are tested.
- The system triggers the first matching rule in the ordered list, and then subsequent matching rules are skipped.
  - If the same rule is configured with both ALLOW and DENY actions, the rule that is defined first in the order is triggered and the subsequent rule (with the other action) is skipped.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.10.1 IP Addresses and Dotted Notation

An IPv4 address consists of four bytes (32 bits). These bytes are also known as octets. To aid with readability, humans typically work with IP addresses in dotted decimal notation. In this notation, there are periods between each of the four numbers (octets) that make up an IP address. For example, an IP address the computers see as:

`00001010 00001010 00001010 00001010`

is written in dotted decimal notation as 10.10.10.10

The mask value as shown in the examples above identifies which of the four bytes (8, 16, 24, 32 bits) the match rule considers when allowing or denying access.

8.5.10.2 vSamples

The mask values in the following IPv4 samples identify which of the four octets (8, 16, 24, 32 bits) the match rule considers when allowing or denying access. The default value is 32. See the mask attribute in the Element reference for more information.

8.5.10.2.1 Deny 10.10.10.10

```
<AccessControl name="ACL">
  <IPRules noRuleMatchAction="ALLOW">
    <MatchRule action="DENY">
      <SourceAddress mask="32">10.10.10.10</SourceAddress>
    </MatchRule>
  </IPRules>
</AccessControl>
```

Deny all requests from client address: 10.10.10.10

Allow requests from any other client address.

8.5.10.2.2 Deny 10.10.10.*

```
<AccessControl name="ACL">
  <IPRules noRuleMatchAction="ALLOW">
    <MatchRule action="DENY">
      <SourceAddress mask="24">10.10.10.10</SourceAddress>
    </MatchRule>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    </IPRules>
</AccessControl>
```

Deny all requests from client address: 10.10.10.*
Allow requests from any other client address.

8.5.10.2.3 Deny 10.10.*.*

```
<AccessControl name="ACL">
  <IPRules noRuleMatchAction="ALLOW">
    <MatchRule action="DENY">
       <SourceAddress mask="16">10.10.10.10</SourceAddress>
    </MatchRule>
  </IPRules>
</AccessControl>
```

Deny all requests from client address: 10.10.*.*
Allow requests from any other client address.

8.5.10.2.4 Deny 10.10.10.*, Allow 10.10.10.20

```
<AccessControl name="ACL">
  <IPRules noRuleMatchAction="ALLOW">
    <MatchRule action="ALLOW">
      <SourceAddress mask="32">10.10.10.20</SourceAddress>
    </MatchRule>
    <MatchRule action="DENY">
      <SourceAddress mask="24">10.10.10.20</SourceAddress>
    </MatchRule>
  </IPRules>
</AccessControl>
```

Deny all requests from client address: 10.10.10.*, but allow 10.10.10.20.
Allow requests from any other client address.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.10.2.5 Allow 10.10.*.*

```
<AccessControl name="ACL">
  <IPRules noRuleMatchAction="DENY">
    <MatchRule action="ALLOW">
      <SourceAddress mask="16">10.10.10.10</SourceAddress>
    </MatchRule>
  </IPRules>
</AccessControl>
```

Allow all requests from address: 10.10.*.*
Deny requests from any other client address.

8.5.10.2.6 Allow Multiple IPs

```
<AccessControl name="ACL">
  <IPRules noRuleMatchAction="DENY">
    <MatchRule action="ALLOW">
      <SourceAddress mask="24">10.10.20.0</SourceAddress>
      <SourceAddress mask="24">10.10.30.0</SourceAddress>
      <SourceAddress mask="24">10.10.40.0</SourceAddress>
     </MatchRule>
  </IPRules>
</AccessControl>
```

Whitelist (allow requests from) client addresses: 10.10.20.* 10.10.30.* 10.10.40.*
Deny all other addresses.

8.5.10.2.7 Deny Multiple IPs

```
<AccessControl name="ACL">
  <IPRules noRuleMatchAction="ALLOW">
    <MatchRule action="DENY">
      <SourceAddress mask="24">10.10.20.0</SourceAddress>
      <SourceAddress mask="24">10.10.30.0</SourceAddress>
      <SourceAddress mask="24">10.10.40.0</SourceAddress>
    </MatchRule>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  </IPRules>
</AccessControl>
```

Blacklist (deny requests from) client addresses: 10.10.20.* 10.10.30.* 10.10.40.*

Allow all other addresses.

8.5.10.2.8 Allow Multiple IPs, Deny Multiple IPs

```
<AccessControl name="ACL">
  <IPRules noRuleMatchAction="DENY">
    <MatchRule action="DENY">
      <SourceAddress mask="24">10.10.0.0</SourceAddress>
      <SourceAddress mask="24">10.20.0.0</SourceAddress>
      <SourceAddress mask="24">10.30.0.0</SourceAddress>
    </MatchRule>
    <MatchRule action="ALLOW">
      <SourceAddress mask="16">10.10.0.0</SourceAddress>
      <SourceAddress mask="16">10.20.0.0</SourceAddress>
      <SourceAddress mask="16">10.30.0.0</SourceAddress>
    </MatchRule>
  </IPRules>
</AccessControl>
```

Whitelist: 10.10.*.* 10.20.*.* 10.30.*.*

Blacklist a subset of the whitelist: 10.10.0.* 10.20.0.* 10.30.0.*

8.5.10.3 Usage notes

In addition to protecting your APIs against malicious IPs, the Access Control policy also gives you control over legitimate IP access. For example, if you only want computers under the control of your enterprise to access the APIs exposed in your test environment, you can allow (or whitelist) the IP address range for your internal network. Developers working from home can access these APIs using VPN.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

> **Note**
>
> If you block a specific address, the computer using that IP address can still direct requests to your API through a proxy. It is also possible to spoof (falsify) the IP address of a request, so the request appears to come from a reputable source.

The configuration and execution of an Access Control policy includes the following tasks and behaviors:

- Define a set of match rules with one of two actions (ALLOW or DENY) associated with each.
- For each match rule, specify the IP address (SourceAddress element).
  - Configure a mask for each IP address. You allow or deny access based on a mask value on the IP address.
  - If the X-FORWARDED-FOR header is present in the request address, then the header value is considered the source address. If there are multiple addresses in the header, use the <ValidateBasedOn> element to control which are evaluated.
- Specify the order in which the rules are tested.
- All the match rules are executed in the given order. When a rules matches, the corresponding action is executed and following match rules are skipped.
  - If the same rule is configured with both ALLOW and DENY actions, the rule that is defined first in the order is triggered and the subsequent rule (with the other action) is skipped.

8.5.10.4 About IP masking with CIDR notation

CIDR notation (Classless Inter-Domain Routing) is a way of indicating a range of IP addresses through masking. It applies to both IPv4 and IPv6. Here's how it works. We'll use IPv4 in our examples for simplicity.

IP addresses are groups of numbers separated by periods. Each number is a specific number of bits (8 for IPv4 and 16 for IPv6). The IPv4 address 10.20.30.40 looks like this in binary:

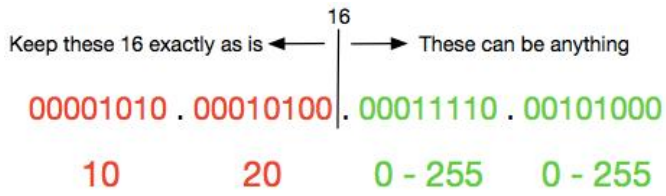
00001010 . 00010100 . 00011110 . 00101000

That's 4 groups of 8 bits, or 32 total bits. With CIDR, you can indicate a range by adding a /number (1-32) to the IP address, like this:

10.20.30.40/16

In this case, the 16 is the number you would use for the mask attribute value in this policy.

This notation means, "Keep the first 16 bits exactly as is, the remaining bits can be anything." For example:



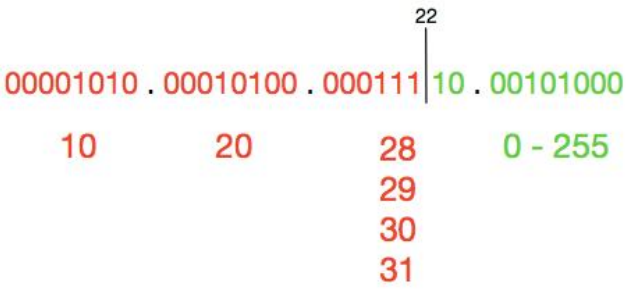
Notice that the mask happens at the end of group two. This makes things nice and tidy, in essence creating a mask like this: 10.20.*.*. In most cases, using multiples of 8 (IPv4) and 16 (IPv6) will give you the masking level you want:

IPv4: 8, 16, 24, 32

IPv6: 16, 32, 48, 64, 80, 96, 112, 128

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

However, you can user other numbers for finer-grained control, which involves a little binary calculation. Here's an example using a mask of 22:



Here it gets a litle tricky: keep the first 22 bits and allow anything in the rest—per group. In group 3, keep the first 6 bits, the last two can vary. That's four possible combinations (00, 01, 10, 11), leaving the following possible numbers in group 3: 28, 29, 30, and 31. Since none of group 4 is affected by the mask, values can be anything from 0 to 255.

8.5.10.5 Element reference

The element reference describes the elements and attributes of the Access Control policy.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AccessControl async="false" continueOnError="false" enabled="true" name="Access-Control-1">
    <DisplayName>Access Control 1</DisplayName>
    <IPRules noRuleMatchAction="ALLOW">
        <MatchRule action="ALLOW">
            <SourceAddress mask="32">10.10.10.20</SourceAddress>
        </MatchRule>
        <MatchRule action="DENY">
            <SourceAddress mask="24">10.10.10.20</SourceAddress>
        </MatchRule>
    </IPRules>
    <ValidateBasedOn>X_FORWARDED_FOR_ALL_IP</ValidateBasedOn>
</AccessControl>
```

8.5.10.6 <AccessControl> attributes

```
<AccessControl async="false" continueOnError="false" enabled="true" name="Access-Control-1">
```

The following attributes are common to all policy parent elements.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Attribute	Description	Default	Presence
name	The internal name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the <DisplayName> element to label the policy in the management UI proxy editor with a different, natural-language name.	NA	Required
continueOnError	Set to false to return an error when a policy fails. This is expected behavior for most policies. Set to true to have flow execution continue even after a policy fails.	False	Optional
enabled	Set to true to enforce the policy. Set to false to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	True	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to true, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to true improves API proxy performance. However, overusing async can hurt performance with too much thread switching.	False	Optional

8.5.10.7 <DisplayName> element

Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

<DisplayName>Policy Display Name</DisplayName>

Default: N/A
If you omit this element, the the value of the policy'sname attribute is used.

Presence: Optional

Type: String

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.10.8 <IPRules> element

The parent element containing the rules that allow or deny IP addresses. The noRuleMatchAction attribute lets you define how to handle any IP addresses that aren't covered by your matching rules.

<IPRules noRuleMatchAction="ALLOW">

Default	N/A
Presence	Optional
Type	N/A

Attributes

Attribute	Description	Type	Default	Presence
noRuleMatchAction	The action to take (allow or deny access) if the match rule specified is not resolved (unmatched). Valid value: ALLOW or DENY	String	ALLOW	Required

8.5.10.9 <IPRules>/<MatchRule> element

The action to take (allow or deny access) if the IP address matches the SourceAddress(es) you define.

```
<IPRules noRuleMatchAction="ALLOW">
    <MatchRule action="ALLOW">
        <SourceAddress mask="32">10.10.10.20</SourceAddress>
    </MatchRule>
    <MatchRule action="DENY">
        <SourceAddress mask="24">10.10.10.20</SourceAddress>
    </MatchRule>
</IPRules>
```

Default	N/A
Presence	Optional
Type	N/A

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Attributes

Attribute	Description	Type	Default	Presence
action	The action to take (allow or deny access) if the match rule specified is not resolved (unmatched). Valid value: ALLOW or DENY	String	ALLOW	Required

8.5.10.10 <IPRules> / <MatchRule> / <SourceAddress> element

The IP address range of a client.

Valid value: Valid IP address (dotted decimal notation). For wildcard behavior, use the mask attribute.

```
<IPRules noRuleMatchAction="ALLOW">
    <MatchRule action="ALLOW">
        <SourceAddress mask="32">10.10.10.20</SourceAddress>
    </MatchRule>
    <MatchRule action="DENY">
        <SourceAddress mask="24">10.10.10.20</SourceAddress>
    </MatchRule>
</IPRules>
```

Default	N/A
Presence	Optional
Type	String

Attributes

Attribute	Description	Type	Default	Presence
mask	The mask attribute is a way to indicate the range of IP addresses to allow or deny. Mask is the equivalent of using CIDR notation (Classless Inter-Domain Routing). For example: <SourceAddressmask="24">192.168.100.0</SourceAddress> is equivalent to the following CIDR notation: 192.168.100.0/24 Valid values:	Integer	N/A	Required

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Attribute	Description	Type	Default	Presence
	IPv4: 1-32 IPv6: 1-128 A value of zero (0) is valid only for IP 0.0.0.0, hence impractical.			

8.5.10.11 <ValidateBasedOn> element

If the X-FORWARDED-FOR HTTP header is present in the request address, then the header value is considered the source address.

If there are multiple IP addresses you want to check in the X-FORWARDED-FOR header—for example, if the header contains the client IP address along with the IP addresses of proxies through which the request was sent (client, proxy1, proxy2)—use this ValidateBasedOn element to control which of the IP addresses are checked.

You must enable this feature, which involves setting the feature.enableMultipleXForwardCheckForACL property in your organization. To enable it, see the API call below.

The value you enter in this element lets you determine whether to check all IP addresses in the header (default), only the first IP address, or only the last IP address.

```
<ValidateBasedOn>X_FORWARDED_FOR_ALL_IP</ValidateBasedOn>
```

Default	X_FORWARDED_FOR_ALL_IP
Presence	Required
Type	String
Valid values	X_FORWARDED_FOR_ALL_IP (default) X_FORWARDED_FOR_FIRST_IP X_FORWARDED_FOR_LAST_IP

You can set the feature.enableMultipleXForwardCheckForACL property with the following API call:

```
curl -u email:password -X POST -H "Content-type:application/xml"
http://host:8080/v1/o/myorg -d \
"<Organization type="trial" name="MyOrganization">
    <DisplayName>MyOrganization</DisplayName>
    <Environments/>
    <Properties>
        <Property name="feature.enableMultipleXForwardCheckForACL">true</Property>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    </Properties>
</Organization>"
```

8.5.11 Minimize API Vunerabilities Using JSON Threat Protection

Like XML-based services, APIs that support JavaScript object notation (JSON) is vulnerable to content-level attacks. Simple JSON attacks attempt to use structures that overwhelm JSON parsers to crash a service and induce application-level denial-of-service attacks.

The JSONThreatProtection policy minimizes the risk posed by such attacks by enabling you to specify limits on various JSON structures, such as arrays and strings. All settings are optional and should be tuned to optimize your service requirements against potential vulnerabilities.

> **Note**
>
> If a limit is not specified, the system applies a default value of -1 (the system equates a negative value to no limit).

8.5.11.1 Configuring the JSON Threat Protection Policy

Configure the JSON Threat Protection policy using the following elements.

> **Note**
>
> The name attribute for this policy is restricted to these characters: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.

Field Name	Description
Source	Request that needs to be validated for JSON payload attacks.
Container depth	Specifies the maximum allowed nested depth. JSON allows you to nest the containers (object and array) in any order to any depth.
Object entry count	Specifies the maximum number of entries allowed in an object.
Object entry name length	Specifies the maximum string length allowed for an object's entry name.
Array element count	Specifies the maximum number of elements allowed in an array.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name	Description
String value length	Specifies the maximum length allowed for a string value.

Example-JSON Threat Protection Policy

```
<JSONThreatProtection name="mypolicy">
    <Source>request</Source>
    <ContainerDepth>10</ContainerDepth>
    <ObjectEntryCount>15</ObjectEntryCount>
    <ArrayElementCount>20</ArrayElementCount>
    <ObjectEntryNameLength>50</ObjectEntryNameLength>
    <StringValueLength>100</StringValueLength>
</JSONThreatProtection>
```

8.5.11.2 Policy-specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

The JSONThreatProtection Policy type defines the following error codes:

Error Code	Message
ExceededContainerDepth	JSONThreatProtection[{0}]: Exceeded container depth at line {1}
ExceededObjectEntryCount	JSONThreatProtection[{0}]: Exceeded object entry count at line {1}
ExceededArrayElementCount	JSONThreatProtection[{0}]: Exceeded array element count at line {1}
ExceededObjectEntryNameLength	JSONThreatProtection[{0}]: Exceeded object entry name length at line {1}
ExceededStringValueLength	JSONThreatProtection[{0}]: Exceeded string value length at line {1}
SourceUnavailable	JSONThreatProtection[{0}]:: Source {1} is notavailable

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Error Code	Message
NonMessageVariable	JSONThreatProtection[{0}]: Variable {1} does notresolve to a Message
ExecutionFailed	JSONThreatProtection[{0}]: Execution failed.reason: {1}

8.5.12 Minimize API Vulnerabilities Using XML Threat Protection

Any server that receives online data is subject to attack, whether malicious or unintentional. Some attacks take advantage of the flexibility of XML by constructing invalid documents that have the potential to compromise back-end systems. Corrupt or extremely complex XML documents can cause servers to allocate more memory than is available, tying up CPU and memory resources, crashing parsers, and generally disabling message processing and creating application-level denial-of-service attacks.

SAP API Management enables you to enforce XMLThreatProtection policies that address XML vulnerabilities and minimize attacks on your API.

You can screen against XML threats using the following approaches:

- Validate messages against an XML schema (.xsd)
- Evaluate message content for specific black-listed keywords or patterns
- Detect corrupt or malformed messages before those messages are parsed

The XMLThreatProtection policy can detect XML payload attacks based on configured limits.

> **Note**
>
> All limits are optional. If a limit is not specified, the system applies a default value of -1 (the system equates a negative value to no limit).

8.5.12.1 Configuring the XMLThreatProtection Policy

Configure the Threat Protection policy using the following elements.

> **Note**
>
> The name attribute for this policy is restricted to these characters: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.

Field Name	Description
Source	Message to be screened for XML payload attacks. This is most commonly set to request, as you will

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name		Description
		typically need to validate inbound requests from client apps. Set this torequest or response. When set to message, this element will automatically evaluate the request message when attached to the request Flow, and the response message when attached to the response Flow.
StructuralLimits (Optional)	NodeDepth	Specifies the maximum node depth allowed in the XML. Valid value: Any integer

	AttributeCountPerElement	Specifies the maximum number of attributes allowed for any element.
		Valid value: Any integer
		Example:
		`<book category="WEB">`
		`<title>Learning XML</title>`
		`<author>Erik T. Ray</author>`
		`<year>2003</year>`
		`</book>`
		The attribute category is checked for the specified limit.
		Note that attributes used for defining namespaces are not counted.

	NamespaceCountPerElement	Specifies the maximum number of namespace definitions allowed in an element.
		Valid value: Any integer
		Example:
		`<e1 attr1="val1" attr2="val2">`
		`<e2 xmlns="http://sap.com" xmlns:yahoo="http://yahoo.com" one="1" yahoo:two="2"/>`
		`</e1>`
		Here <e1> has 0 namespace definitions and <e2> has 2 namespace definitions. Note that attributes used for defining namespaces are not counted.

	ChildCount	Specifies a limit on the maximum number of children allowed for any element in the XML documemnt. Valid value: Any integer

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name		Description

ValueLimits(Optional)	Text	Specifies a limit on the maximum length, in characters, of any text nodes present in the XML document.
		Valid value: Any integer
		Example:
		`<book category="WEB">`
		`<title>Learning XML</title>`
		`<author>Erik T. Ray</author>`
		`<year>2003</year>`
		`</book>`
		The text nodes Learning XML, Erik T. Ray, and2003 are checked for the specified limit.

	Attribute	Specifies a limit on the maximum length, in characters, of any attributes for any element in the XML document.
		Valid value: Any integer
		Example:
		`<book category="WEB">`
		`<title>Learning XML</title>`
		`<author>Erik T. Ray</author>`
		`<year>2003</year>`
		`</book>`
		The attribute node category is checked for the specified limit.

	NamespaceURI	Specifies a limit on the maximum number of namespaces in the XML document. Valid value: Any integer Example: <ns1:myelem xmlns:ns1="http://ns1.com"/> The namespace value http://ns1.com is checked for the specified limit.

	Comment	Specifies a limit on the maximum number of comment characters in the XML document.
		Valid value: Any integer
		Example:
		`<book category="WEB">`
		`<!-- This is a comment -->`
		`<title>Learning XML</title>`
		`<author>Erik T. Ray</author>`
		`<year>2003</year>`

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name		Description
		`</book>` The comment text <!-- This is a comment --> is checked for the specified limit.
	ProcessingInstructionData	Specifies a limit on the maximum number of characters for any processing instructions content in the XML document. Valid value: Any integer Example: <?xml-stylesheet type="text/xsl" href="style.xsl"?> The value of processing instruction contenttype="text/xsl" href="style.xsl" is checked for the specified limit.

NameLimits (Optional)	Element	Specifies a limit on the maximum number of characters permitted in any element name in the XML document.
		Valid value: Any integer
		Example:
		`<book category="WEB">`
		`<title>Learning XML</title>`
		`<author>Erik T. Ray</author>`
		`<year>2003</year>`
		`</book>`
		The elements `title`, `author`, and `year` are checked for the specified limit.

	Attribute	Specifies a limit on the maximum number of characters in any name for any attribute in the XML document.
		Valid value: Any integer
		Example:
		`<book category="WEB">`
		`<title>Learning XML</title>`
		`<author>Erik T. Ray</author>`
		`<year>2003</year>`
		`</book>`
		The attribute `category` is checked for the specified limit.

	NamespacePrefix	Specifies a limit on the maximum number of characters in the namespace prefix. Valid value: Any integer

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name		Description
		Example: `<ns1:myelem xmlns:ns1="http://ns1.com"/>` The prefix `ns1` is check for the specified limit.
	ProcessingInstructionTarget	Specifies a limit on the maximum number of characters in the target of any processing instructions in the XML document. Valid value: Any integer Example: `<?xml-stylesheet type="text/xsl" href="style.xsl"?>` The value of processing instruction target `xml-stylesheet` is checked for the specified limit.

Example-XMLThreatProtection policy

```
<XMLThreatProtection name="mypolicy">
    <Source>request</Source>
    <StructureLimits>
        <NodeDepth>5</NodeDepth>
        <AttributeCountPerElement>3</AttributeCountPerElement>
        <NamespaceCountPerElement>2</NamespaceCountPerElement>
        <ChildCount includeText="true"
                    includeComment="true"
                    includeProcessingInstruction="true"
                    includeElement="true">3</ChildCount>
    </StructureLimits>
    <ValueLimits>
         <Text>15</Text>
         <Attribute>10</Attribute>
         <Namespace>10</Namespace>
         <Comment>10</Comment>
         <ProcessingInstructionData>10</ProcessingInstructionData>
    </ValueLimits>
    <NameLimits>
          <Element>10</Element>
          <Attribute>10</Attribute>
          <Prefix>10</Prefix>
          <ProcessingInstructionTarget>10</ProcessingInstructionTarget>
    </NameLimits>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
</XMLThreatProtection>
```

8.5.12.2 Policy-specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

The XMLThreatProtection Policy type defines the following error codes:

Error Code	Message
NodeDepthExceeded	XMLThreatProtection stepDefinition {0}: Node depth exceeded{1}
AttrCountExceeded	XMLThreatProtection stepDefinition {0}: Attribute count exceeded {1}
ChildCountExceeded	XMLThreatProtection stepDefinition {0}: Children count exceeded {1}
NSCountExceeded	XMLThreatProtection stepDefinition {0}: Namespace count exceeded {1}
ElemNameExceeded	XMLThreatProtection stepDefinition {0}: Element name length exceeded {1}
AttrNameExceeded	XMLThreatProtection stepDefinition {0}: Attribute name length exceeded {1}
AttrValueExceeded	XMLThreatProtection stepDefinition {0}: Attribute value length exceeded {1}
NSPrefixExceeded	XMLThreatProtection stepDefinition {0}: Namespace prefix length exceeded {1}
NSURIExceeded	XMLThreatProtection stepDefinition {0}: Namespace uri length exceeded {1}
PITargetExceeded	XMLThreatProtection stepDefinition {0}: Processing Instructiontarget length exceeded {1}
PIDataExceeded	XMLThreatProtection stepDefinition {0}: Processing Instructiondata length exceeded {1}
CommentExceeded	XMLThreatProtection stepDefinition {0}: Comment length exceeded {1}
TextExceeded	XMLThreatProtection stepDefinition {0}: Text length exceeded{1}
SourceUnavailable	XMLThreatProtection stepDefinition {0}: Source {1} is notavailable
NonMessageVariable	Variable {0} does not resolve to a Message

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Error Code	Message
ExecutionFailed	XMLThreatProtection stepDefinition {0}: Execution failed.reason: {1}

8.5.13 Search Threat Patterns Using Regular Expression

8.5.13.1 Evaluate message content using RegularExpressionProtection

A regular expression, or regex for short, is a set of strings that specify a pattern in a string. Regular expressions enable content to be programmatically evaluated for patterns. Regular expressions can be used, for example, to evaluate entered a properly structured email address.

SAP API Management enables you to configure regular expressions that can be evaluated at runtime against API traffic to identify common content-level threats that follow certain patterns. The policy extracts information from a message (for example, URI Path, Query Param, Header, Form Param, Variable, XML Payload, or JSON Payload) and evaluates that content against pre-defined regular expressions. If any specified regular expressions evaluate to true, the message is considered a threat and is rejected.

The most common usage of RegularExpressionProtection is the evaluation of JSON and XML payloads for malicious content.

For example, to evaluate JSON payloads for SQL injection attacks, evaluate the JSON payload using the following JSONPath expression combined with a regular expression. The regular expression pattern is evaluated against the content extracted by the JSONPath expression.

Example

You need to URL encode regular expressions in the XML configuration file. In the example below, the following regular expression:

```
[\s]*((delete)|(exec)|(drop\s*table)|(insert)|(shutdown)|(update)|(\bor\b))
```

is encoded as:

```
%5B%5Cs%5D%2A%28%28delete%29%7C%28exec%29%7C%28drop%5Cs%2Atable%29%7C%28insert%29%7C%28shutdown%29%7C%28update%29%7C%28%5Cbor%5Cb%29%29
```

Example

<Put your example here>

```
<RegularExpressionProtection name="JsonSQLInjectionEvaluation">
  <Source>request</Source>
  <JSONPayload>
    <JSONPath>
      <Expression>$.</Expression>
      <--The regular expression below is URL encoded -->
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<Pattern>%5B%5Cs%5D%2A%28%28delete%29%7C%28exec%29%7C%28drop%5Cs%2Atable%29%7C%28inser
t%29%7C%28shutdown%29%7C%28update%29%7C%28%5Cbor%5Cb%29%29</Pattern>
    </JSONPath >
  </JSONPayload>
</RegularExpressionProtection>
```

No regular expression can eliminate all content-based attacks, and multiple mechanisms should be combined to enable defense-in-depth. With this in mind, some recommended patterns for blacklisting content:

8.5.13.1.1 Blacklist Patterns

Regular expressions must be URL encoded in the policy's XML configuration file.

Name	Regular Expression
SQL Injection	`[\s]*((delete)|(exec)|(drop\s*table)|(insert)|(shutdown)|(update)|(\bor\b))`
Server-Side Include Injection	`<!--\s*<!--(include|exec|echo|config|printenv)\s+.*`
XPath Abbreviated Syntax Injection	`(/(@?[\w_?\w:\*]+(\+\])*)?)+`
XPath Expanded Syntax Injection	`/?(ancestor(-or-self)?|descendant(-or-self)?|following(-sibling))`
JavaScript Injection	`<\s*script\b[^>]*>[^<]+<\s*/\s*script\s*>`
Java Exception Injection	`.*Exception in thread.*`

8.5.13.1.2 Configuring RegularExpressionProtection Policy

Configure the RegularExpressionProtection policy using the following elements.

> **Note**
>
> The name attribute for this policy is restricted to these characters: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.

Field Name	Description
Source (Optional)	Contains the message from which information needs to be extracted. · If the Source element is omitted, the value defaults to message. For example, <Source>message</Source>.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name			Description
			When set to message, the policy uses the request message as source when attached to a request Flow. Likewise, the policy uses the response message when attached to a response Flow. · If Source cannot be resolved, or if it resolves to a non-message type, the policy returns an error.
IgnoreUnresolvedVariables (Optional)			Valid values: true/false Default value: false If set to true and any variable is unresolvable, the policy returns an error. If set to true and any variable is unresolvable, the unresolved variable is treated as empty string (Null).
Patterns (Optional)	URIPath		Extracts information from the request URI path and matches it with the specified regular expressions.
	QueryParam		Extracts information from the request query parameter and matches it with the specified regular expressions.
	Header		Extracts information from the request and response header and matches it with the specified regular expressions.
	FormParam		Extracts information from the request form parameter and matches it with the specified regular expressions.
	Variable		Extracts information from the given variable and matches it with the specified regular expressions.
XMLPayload (Optional)	Namespaces		Specifies the namespace to be used in the XPath evaluation.
	XPath	Expression	Specifies the XPath expression defined for the variable. Only XPath 1.0 expressions are supported. For example,<Expression>/company/employee[@age>=$request.header.age]</Expression> extracts details for employees whose age is greater than or equal to the value specified in request.header.age.
		Type	Specifies the datatype; default is string.
		Pattern	Defines the regular expression pattern.
JSONPayload (Optional)	JSONPath	Expression	Specifies the JSONPath expression defined for the variable.
		Pattern	Defines the regular expression pattern.

Example-RegularExpression Protection policy

```
<RegularExpressionProtection name="mypolicy">
    <Source>response</Source>
    <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <URIPath>
        <Pattern>[tT]rue</Pattern>
        <Pattern>.*true.*</Pattern>
    </URIPath>
    <QueryParam name="greeting">
        <Pattern>[tT]rue</Pattern>
        <Pattern>.*true.*</Pattern>
    </QueryParam>
    <Header name="greeting">
        <Pattern>[tT]rue</Pattern>
        <Pattern>.*true.*</Pattern>
    </Header>
    <FormParam name="greeting">
        <Pattern>[tT]rue</Pattern>
        <Pattern>.*true.*</Pattern>
    </FormParam>
    <Variable name="request.content">
        <Pattern>[tT]rue</Pattern>
        <Pattern>.*true.*</Pattern>
    </Variable>
    <XMLPayload>
        <Namespaces>
            <Namespace prefix="sap">http://www.sap.com</Namespace>
        </Namespaces>
        <XPath>
            <Expression>/sap:Greeting/sap:User</Expression>
            <Type>string</Type>
            <Pattern>[tT]rue</Pattern>
            <Pattern>.*true.*</Pattern>
        </XPath>
    </XMLPayload>
    <JSONPayload>
        <JSONPath>
            <Expression>$.store.book[*].author</Expression>
            <Pattern>[tT]rue</Pattern>
            <Pattern>.*true.*</Pattern>
        </JSONPath >
    </JSONPayload>
</RegularExpressionProtection>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.13.1.3 Policy-specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

The RegularExpressionProtection Policy type defines the following error codes:

Error Code	Message
NothingToEnforce	RegularExpressionProtection {0}: at least one ofURIPath, QueryParam, Header, FormParam,XMLPayload, JSONPayload is mandatory
NoPatternsToEnforce	RegularExpressionProtection {0}: No patterns to enforce in {1}
EmptyXPathExpression	RegularExpressionProtection {0}: Empty XPathexpression
EmptyJSONPathExpression	RegularExpressionProtection {0}: Empty JSONPathexpression
DuplicatePrefix	RegularExpressionProtection {0}: Duplicate prefix{1}
NONEmptyPrefixMappedToEmptyURI	RegularExpressionProtection {0}: Non-empty prefix{1} cannot be mapped to empty uri
ThreatDetected	Regular Expression Threat Detected in {0}: regex:{1} input : {2}
ExecutionFailed	Failed to execute the RegularExpressionProtectionStepDefinition {0}. Reason: {1}
VariableResolutionFailed	Failed to resolve variable {0}
NonMessageVariable	Variable {0} does not resolve to a Message
SourceMessageNotAvailable	{0} message is not available forRegularExpressionProtection StepDefinition {1}
InvalidRegularExpression	RegularExpressionProtection {0}: Invalid RegularExpression {1}, Context {2}
InstantiationFailed	Failed to instantiate theRegularExpressionProtection StepDefinition {0}
CannotBeConvertedToNodeset	RegularExpressionProtection {0}: Result of xpath{1} cannot be converted to nodeset. Context {2}
XPathCompilationFailed	RegularExpressionProtection {0}: Failed to compile xpath {1}. Context {2}

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Error Code	Message
JSONPathCompilationFailed	RegularExpressionProtection {0}: Failed to compile jsonpath {1}. Context {2}

8.5.14 Verify API Keys Using API Key Validation

8.5.14.1 Enforce Access Control Using VerifyAPIKey

RESTful services are often exposed over the open Internet for consumption by developers building apps. API providers need a mechanism to prevent unauthorized access to the API. One approach is to provision developers with API keys and secrets. Those keys can be used as authentication tokens, or they can be used to obtain OAUth access tokens.

SAP API Management automatically generates API keys on behalf of apps. It enables API providers to view, approve, and revoke API keys. By applying a policy of the type VerifyApiKey, you can enforce verification of API keys at runtime. This ensures that no app can access a protected API without a valid key.

API keys are verified at runtime to ensure that:

- The API key presented is valid and has not been revoked
- The API key presented is approved to consume an API product that includes the URI in the request

The only setting required for the VerifyAPIKey defines the expected location of the API key. Only one key location is supported per policy instance.

For instructions on configuring API key validation locally, refer to API keys.

> **Note**
>
> "API keys" are also referred to as "consumer keys" and "app keys". In OAuth, API keys are referred to as 'client id'. The names can be used interchangeably.

8.5.14.1.1 Samples

Key in Query Param

```
<VerifyAPIKey name="APIKeyVerifier">
    <APIKey ref="request.queryparam.apikey" />
</VerifyAPIKey>
```

The policy extracts the API key from the request message by referencing the flow variable specified by the ref attribute. In this example, the policy extracts the API key from a flow variable named request.queryparam.apikey, which is populated by a query param named apikey.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The following URL shows how you pass the API key for this example:

```
http://{org-name}-
{env}.<host:port>/v1/weather/forecastrss?w=12797282&apikey=IEYRtW2cb7A5Gs54A1wKElECBL6
5GVls
```

where `{org-name}` is the name of your SAP API Management organization.

Key in header

```
<VerifyAPIKey name="APIKeyVerifier">
    <APIKey ref="request.header.AppKey" />
</VerifyAPIKey>
```

In this example, you configure the policy to look for the API key in a header named AppKey. The client app must then pass the API key as the value of an HTTP header named AppKey.

Key in Variable

```
<VerifyAPIKey name="APIKeyVerifier">
    <APIKey ref="requestAPIKey.key"/>
</VerifyAPIKey>
```

The policy can reference any variable that contains the key. The policy in this example extracts the API key from a variable namedrequestAPIKey.key.

How that variable is populated is up to you. For example, you could use the Extract Variables policy to populate requestAPIKey.key from a query parameter named myKey, as shown below:

```
<ExtractVariables async="false" continueOnError="false" enabled="true"
name="SetAPIKeyVar">
    <Source>request</Source>
    <QueryParam name="myKey">
        <Pattern ignoreCase="true">{key}</Pattern>
    </QueryParam>
    <VariablePrefix>requestAPIKey</VariablePrefix>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</ExtractVariables>
```

Access Policy Flow Variables

```
<AssignMessage async="false" continueOnError="false" enabled="true"
name="accessverifyvars">
    <AssignVariable>
        <Name>devFirstName</Name>
        <Ref>verifyapikey.verify-api-key.developer.firstName</Ref>
        <Value>ErrorOnCopy</Value>
    </AssignVariable>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <AssignVariable>
        <Name>devLastName</Name>
        <Ref>verifyapikey.verify-api-key.developer.lastName</Ref>
        <Value>ErrorOnCopy</Value>
    </AssignVariable>
    <AssignVariable>
        <Name>devEmail</Name>
        <Ref>verifyapikey.verify-api-key.developer.email</Ref>
        <Value>ErrorOnCopy</Value>
    </AssignVariable>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
    <AssignTo createNew="false" transport="http" type="request"/>
</AssignMessage>
```

SAP API Mangement automatically populates a set of flow variables when executing the Verify API Key policy for a valid API key. You can use these variables to access information such as the app name, app ID, and information about the developer or company who registered the app. In the example above, you use the Assign Message policy to access the developer's first name, last name, and email address after the Verify API Key executes.

These variables are all prefixed by:

verifyapikey.{policy_name}

In this example, the Verify API key policy name is "verify-api-key". Therefore, you reference the first name of the developer making the request by accessing the variable verifyapikey.verify-api-key.developer.firstName.

For a list of available request message variables, see Variables reference.

8.5.14.1.2 Element Reference

Following are elements and attributes you can configure on this policy.

```
<VerifyAPIKey async="false" continueOnError="false" enabled="true" name="Verify-API-Key-1">
    <DisplayName>Custom label used in UI</DisplayName>
    <APIKey ref="variable_containing_api_key"/>
</VerifyAPIKey>
```

<VerifyAPIKey> attributes

```
<VerifyAPIKey async="false" continueOnError="false" enabled="true" name="Verify-API-Key-1">
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The following attributes are common to all policy parent elements.

Attribute	Description	Default	Presence
name	The internal name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the <DisplayName> element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Required
continueOnError	Set to false to return an error when a policy fails. This is expected behavior for most policies. Set to true to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to true to enforce the policy. Set to false to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to true, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to true improves API proxy performance. However, overusing async can hurt performance with too much thread switching. To use asynchronous behavior in API proxies, see JavaScript callouts.	false	Optional

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.15 Authenticating Users and Getting User Distinguished Names in LDAP Policy

> **Note**
>
> This topic pertains to SAP API Management SAP API Management on-premises only.

SAP API Management on-premise lets you leverage an LDAP provider in API calls. With the LDAP Policy, applications can authenticate credentials against users stored in LDAP, and you can retrieve distinguished names (DNs) from LDAP—the metadata, or attributes, associated with each user, such as email, address, and phone number. The returned DN is stored in a variable for further use by the API proxy.

The LDAP Policy provides:

- Authentication: User credentials supplied in the request are validated against credentials in the LDAP provider. The LDAP policy gives you a lot of flexibility with authentication, letting you use any DN value along with the password, even if that DN value you want isn't in the request. For example, say you need to use email password for authentication. The following options are possible:
  - If the email is in the request, you can simply use that with the password for LDAP authentication.
  - If the email isn't in the request, but another DN attribute is (such as phone number), you can use the phone number to get the corresponding email from LDAP, then use email / password to authenticate.
- Distinguished name (DN) search: In addition to authentication, you can also use the LDAP Policy to identify a user attribute in the request, such as email, and perform a query that retrieves other DN attributes from LDAP for that user. The retrieved DN is stored in a variable.

8.5.15.1 Use case

Use the LDAP Policy when access to protected resources should be limited to users in your LDAP provider—such as your admin users, organization users, and developers—especially when OAuth token access is either unnecessary or too heavyweight. The policy is also designed for retrieving DN metadata for use in API proxy flows.

> **Example**
>
> You can have an API call execute only when a user is successfully authenticated against LDAP; and then optionally retrieve DN attributes for the user after authentication succeeds.

8.5.15.2 Samples

Following are examples of using the LDAP Policy for authentication and DN search. See the “Policy settings reference” and “Policy-specific variables” sections below for more information on these examples.

Authentication Against an LDAP Provider

This policy passes username and password from the request to LDAP for authentication.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<Ldap name="4GLdapPolicy">
   <LdapResource>ldap1</LdapResource>
   <Authentication>
       <UserName ref=”request.header.username”/>
       <Password ref=”request.header.password”/>
       <Scope>subtree</Scope>
       <BaseDN></BaseDN>
    </Authentication>
 </Ldap>
```

Authentication Against an LDAP Provider Using a DN Attribute Other Than Username

This policy gets the user's DN with the email in the request header and then authenticates the user against LDAP with the password provided in the request header.

```
<Ldap name="4GLdapPolicy">
   <LdapResource>ldap1</LdapResource>
   <Authentication>
       <Password ref=”request.header.password”/>
       <SearchQuery>mail={request.header.mail}</SearchQuery>
       <Scope>subtree</Scope>
       <BaseDN></BaseDN>
    </Authentication>
 </Ldap>
```

> **Note**
>
> Notice that in the Authentication block there is no UserName, which is not allowed when using SearchQuery to retrieve another DN attribute for authentication.

Searching LDAP - custom LDAP provider

This policy references a custom LDAP provider. It uses the email address in the request header to identify the user, then retrieves the user's address, phone, and title from LDAP. The retrieved DN attributes are stored in a variable. See "Policy-specific variables".

To search LDAP and retrieve DN attributes, the request must include administrator credentials.

```
<Ldap name="4GLdapPolicy">
    <!-- using a custom LDAP provider -->
    <LdapConnectorClass>com.mycompany.MyLdapProvider</LdapConnectorClass>
    <LdapResource>MyLdap</LdapResource>
    <Search>
        <BaseDN></BaseDN>
        <SearchQuery>mail={request.header.mail}</SearchQuery>
        <Attributes>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
            <Attribute>address</Attribute>
            <Attribute>phone</Attribute>
            <Attribute>title</Attribute>
        </Attributes>
        <Scope></Scope> <!-- default is ‘subtree’ -->
    </Search>
</Ldap>
```

8.5.15.3 Response Codes

Following are the HTML response codes the policy returns for success or failure:

- Success: 200
- Failure: 401

8.5.15.4 Policy Settings Reference

Following are descriptions of the LDAP Policy elements and attributes.

> **Note**
>
> The name attribute for this policy is restricted to these characters: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.

Element	Description
Ldap	Parent element with a name attribute for you to enter the policy name.
LdapConnectorClass	When using the LDAP Policy with a custom LDAP server (not provided by SAP API Management), specify the fully qualified LDAP connector class. That’s the class in which you implemented SAP API Management ExternalLdapConProvider interface.
LdapResource	Enter the environment name of the LDAP resource (you must create this resource in your environment).
BaseDN	The base level of LDAP under which all of your data exists.
Scope	object: Authentication or search occurs only at the base level of LDAP. onelevel: Authentication or search occurs one level below the base level. subtree (default): Authentication or search occurs at the base level and fully recursively below base.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Element	Description
Authentication	
Authentication	Parent element for the authentication behavior you implement.
UserName	Empty element that takes one of the following attributes: · ref: A reference to the username in the request, such as request.header.username · value: The username itself If you aren't authenticating with username, or if username isn't included in the request, you don't need to include this element. If username is in the request, but you want to authenticate a user with a DN attribute other than username, such as email, include a SearchQuery to get the user email associated with the password. The LDAP policy uses username to query the LDAP provider for the corresponding email address, which is then used for authentication.
Password	Empty element that takes one of the following attributes: · ref: A reference to the password in the request, such as request.header.password · value: The encrypted password itself
SearchQuery	If you want to authenticate using a DN attribute other than username, such as email, configure the LDAP policy to get a DN attribute from the request (such as username), which is used to identify the user in LDAP, retrieve the email, and authenticate the user. For example, assuming LDAP defines a "mail" attribute for storing email address: `<SearchQuery>mail={request.header.mail}</SearchQuery>`
Search	
Search	Parent element for the search behavior you implement.
SearchQuery	By identifying the user with metadata in the request or response, you can use this element to retrieve additional DN attributes for the user from LDAP. For example, if the request contains the user email, and your LDAP defines a “mail” attribute for storing user email addresses, you’d use the following setting: <SearchQuery>mail={request.header.mail}</SearchQuery> This query searches LDAP for an email matching the email in the request, and the policy can now retrieve additional DN attributes for that user with the Attributes element.
Attributes	Use one or more <Attribute> elements to identify the DN metadata you want to retrieve for the user. At least one attribute is required.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Element	Description

	For example, after the SearchQuery identifies the user, the policy can now retrieve DN attributes for the user such as address, phone number, and the user's title, as shown in the following example.
	Attribute values are the DN attribute names defined in your LDAP.
	`<Attributes>`
	`<Attribute>address</Attribute>`
	`<Attribute>phone</Attribute>`
	`<Attribute>title</Attribute>`
	`</Attributes>`

8.5.15.5 Policy-Specific Variables

Following are the LDAP Policy variables populated by a `SearchQuery`.

Variable	Description
`ldap.<policyName>.execution.success`	After the policy is executed, this flow variable contains a value of true or false depending on the result.
`ldap.<policyName>.search.result[index].attribute.<attrName>[index]=<value>`	The flexible format of this variable—the "index" in particular—accounts for multiple attributes, as well as attributes with multiple values. Index is a number that starts at 1. If no index number is provided, the default index number is 1. If the policy returns address, phone, and email, you can retrieve the first attribute and value using these variables: ldap.<policyName>.search.result.attribute.address ldap.<policyName>.search.result.attribute.phone ldap.<policyName>.search.result.attribute.email If you wanted to retrieve the third address attribute in the search results, you'd use this: ldap.<policyName>.search.result[3].attribute.address If an attribute had multiple values (for example, if a user has multiple email addresses), you'd retrieve the second email address from the results like this: ldap.<policyName>.search.result.attribute.mail[2]

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.5.15.6 Policy-Specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

Error Code	Message
InvalidAttributeName	Invalid attribute name {0}.
InvalidSearchBase	Search base can not be empty.
InvalidValueForPassword	Invalid value for password field. It can notbe empty.
InvalidSearchScope	Invalid scope {0}. Allowed scopes are {1}.
InvalidUserCredentials	Invalid user credentials.
InvalidExternalLdapReference	Invalid external ldap reference {0}.
LdapResourceNotFound	Ldap resource {0} not found.
BaseDNRequired	Base DN required.
OnlyReferenceOrValueIsAllowed	Only value or reference is allowed for {0}.
AttributesRequired	At least one attribute required for search action.
UserNameIsNull	User name is null.
SearchQueryAndUserNameCannotBePresent	Both search query and username can not be present in the authentication action. Please specify either one of them.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.6 Extension Policy

8.6.1 Call services or APIs using ServiceCallout

APIs are at their best when they are combined to create composite services that bundle resources from multiple services and API providers. One way of bundling API calls is to configure a ServiceCallout policy within a request or response flow.

A typical use case involves a service callout from a response flow to a third-party API (external service). The response from the third-party API is parsed and inserted in the response message to the requesting app. The data returned from the backend service is enriched with the data obtained from the third-party API, providing the end users of the app with data 'mashed up' from multiple services. Using a ServiceCallout policy, you can configure an API proxy to make multiple API calls to deliver relevant customer reviews, items from a partner's retail catalog, and so on.

(An alternative approach to ServiceCallout is an HTTP Client written in JavaScript using the JavaScript object model.

8.6.1.1 Policy Composition

Policy composition is an important concept to understand when configuring ServiceCallouts. Policy composition is the definition of a sequence of policies that work together to perform a task. The ServiceCallout is usually used with two other policy types, AssignMessage and ExtractVariables. The AssignMessage policy is used to populate the request message sent to the remote service. The ExtractVariables policy is used to parse the response and to extract specific content.

The typical ServiceCallout policy composition involves:

1. AssignMessage Policy: Creates a request message, populates HTTP headers, query parameters, sets the HTTP verb, etc.
2. ServiceCallout Policy: References a message created by the Assign Message policy, defines a target URL for the external call, and defines a name for the response object that the target service returns.
3. ExtractVariables Policy: Typically defines a JSONPath or XPath expression that parses the message generated by the preceding ServiceCallout policy. The policy then sets variables containing the values parsed from the ServiceCallout response.

For more information, see Generate or modify messages using AssignMessage and Extract message content using ExtractVariables.

8.6.1.2 Sample

The following ServiceCallout policy submits a request to the Google Geocoding API. The content of the request message is extracted from a variable called GeocodingRequest (which could be populated, for example, by an AssignMessage policy). The response message is assigned to the variable called GeocodingResponse, where it is available to be parsed, for example, by an ExtractVariables policy or by custom code written in JavaScript.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<ServiceCallout name="GeoCodeClient">
    <Request clearPayload="false" variable="GeocodingRequest" />
    <Response>GeocodingResponse</Response>
    <Timeout>30000</Timeout>
    <HTTPTargetConnection>
      <URL>http://maps.googleapis.com/maps/api/geocode/json</URL>
    </HTTPTargetConnection>
</ServiceCallout>
```

8.6.1.3 Configuring the ServiceCallout Policy

Configure the ServiceCallout policy using the following elements.

> **Note**
>
> The name attribute for this policy is restricted to these characters: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.

Field Name	Descripti
Request (Optional)	The variable that contains the request message to be sent by the ServiceCallout. · By default, clearPayload is false. · If clearPayload is set to true, the request payload is cleared after the request is sent to the HTTP target. · Use the clearPayload option only if the request message is not required after the ServiceCallout is executed, because clearPayload allocates memory during message processing. · The policy returns an error if the request message cannot be resolved by the element or is of an invalid request message type.
Response (Optional)	Output of the ServiceCallout (usually the response message received from the target) that will be assigned to the response variable. The output generated by the policy is assigned to the variable only when the entire response is read successfully by the policy. If the response message fails for any reason, the policy returns an error. If this element is not specified, the policy execution does not wait for response to be completely read and executes the message flow steps.
Timeout (Optional)	The time in milliseconds that the ServiceCallout policy will wait for a response from the target before exiting. The default timeout for ServiceCallout is determined by the default HTTP timeout setting for SAP API Management, which is 120000 milliseconds (120 seconds).
HTTPTargetConnection	Provides transport details such as URL, SSL, HTTP properties. See theTargetEndpoint configuration reference.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name	Descripti
	Note You can use flow variables to construct the URL in an HttpTargetConnection element. See the example below.

Example

ServiceCallout policy

```
<ServiceCallout name="myPolicy">
    <Request clearPayload="false" variable="myRequest"/>
    <Response>myResponse</Response>
    <HTTPTargetConnection>
        <Properties/>
        <URL>http://example.com</URL>
    </HTTPTargetConnection>
</ServiceCallout>
```

Example

Using variables in a URL

```
<HTTPTargetConnection>
    <Properties/>
    <URL>http://${request.header.targethost}/forecastrss?w=2442047</URL>
</HTTPTargetConnection>
```

8.6.1.4 Policy-specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

The ServiceCallout Policy type defines the following error codes:

Error Codes	Message
ConnectionInfoMissing	Connection information is missing in Step{0}
ExecutionFailed	Execution of ServiceCallout {0} failed.Reason: {1}
RequestVariableIsNull	ServiceCallout[{0}]: request variable {1}value is null

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Error Codes	Message
RequestVariableNotMessageType	ServiceCallout[{0}]: request variable {1}value is not of type Message
RequestVariableNotRequestMessageType	ServiceCallout[{0}]: request variable {1}value is not of type Request Message
ErrorResponseCode	ResponseCode {0} is treated as error
InvalidTimeoutValue	Invalid Timeout value {0}
URLMissing	JSONToXML[{0}]: Output variable is notavailable.

8.6.2 Capture Analytics from API Traffic

The StatisticsCollector policy type enables you to collect statistical data for messages processed in a flow, such as product ID, price, REST action, client and target URL, and message length (predefined flow variables as well as custom variables). The data is then provided to the analytics server, which analyzes the statistics and generates reports. You can view the reports through Analytics Services.

You attach the StatisticsCollector Policy to the response path in your API proxy, where you can capture and publish a range of statistics to the analytics server.

For more information on how to analyze your data using analytics services, see Analyze API message content using custom analytics.

8.6.2.1 Configuring a Statistics Collector Policy

Configure the Statistics Collector Policy using the following elements.

Field Name	Description
name	Custom name provided in the referenced variable. The name cannot contain spaces or dashes. In the example below, the referenced variable is product.id. **Note** The following restrictions apply: · Names cannot be multiple-word (no spaces). · No quotation marks, underscores, hyphens, or periods. · No special characters. · In addition to the above, follow the column naming restrictions here: http://www.postgresql.org/docs/8.0/static/sql-syntax.html#SQL-SYNTAX-IDENTIFIERS

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name	Description
Ref	Flow variable that can be a predefined variable or a variable defined in the XML payload of the Extract Variables policy. See Extract message content using ExtractVariables.
type	Valid values: string/integer/float/long/boolean/double If provided, overrides the existing datatype of ref. Note that the type can be left empty only if ref is a predefined variable or the type is declared in the XML payload of the Extract Variables policy.

You can set the defaultval as shown in the following example to initialize a variable with a static, default value. If you provide a default value, then the default value you provide will be assigned to the variable the variable only when the variable does not get initialized during the message flow. If you do not provide a defaultval, then the variable is assigned normally during the message flow and calls fail if the variable is not resolved.

Example

```
<StatisticsCollector name="publishPurchaseDetails">
<Statistics>
<Statistic name="productID" ref="product.id" type="string">{defaultval}</Statistic>
<Statistic name="price" ref="product.price" type="string">{defaultval}</Statistic>
</Statistics>
</StatisticsCollector>
```

8.6.2.2 Policy-specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

The StatisticsCollector Policy type defines the following error codes:

Error Code	Message
DatatypeMissing	StatisticsCollection {0}: Datatype of {1} is missing. Context{2}
UnsupportedDatatype	StatisticsCollection {0}: Datatype {1} is unsupported .Context {2}

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.6.3 Customize an API Using JavaScript

SAP API Management provides a range of out-of-the-box policy capabilities that address common API management requirements. However, there are some cases where your API requires custom behavior that is not covered by SAP API Management standard policy palette. To support these requirements, SAP API Management exposes scripting interfaces to ease the task of implementing custom behaviors in the proxied API message flow. One approach is to attach your own JavaScript to an API flow, which SAP API Management then executes at runtime.

There are many use cases for the JavaScript policy. For example, you can get and set flow variables, execute custom logic and perform fault handling, extract data from requests or responses, dynamically edit the backend target URL, and much more. This policy allows you to implement custom behavior that is not covered by any other standard SAP API Management policies. In fact, you can use a JavaScript policy to achieve many of the same behaviors implemented by other policies, like AssignMessage and ExtractVariable.

Basically, the JavaScript policy specifies the name of the JavaScript source file to execute when the step to which the policy is attached executes. The source file is always stored in a standard location within the proxy bundle: apiproxy/resources/jsc. Or, you can also store the source code in a resource file at the environment or organization level. You can also upload your JavaScript through the management UI proxy editor.

JavaScript source files must always have a .js extension.

8.6.3.1 Samples

8.6.3.1.1 Rewriting the Target URL

Here's a common use case: extracting data from a request body, storing it in a flow variable, and using that flow variable elsewhere in the proxy flow. Let's say you have a weather app where the user enters their zip code in an HTML form and submits it. You want the API proxy to extract the form data and dynamically add it to the URL used to call the backend service. How would you do this in a JavsScript policy?

> **Note**
>
> If you want to try out this example, we assume you've created a new proxy in the proxy editor. When you create it, just give it a backend service URL of: http://www.example.com. For this example, we're going to rewrite the backend URL dynamically. If you don't know how to create a new proxy, refer to the getting started tutorial.

```
if (context.flow=="PROXY_REQ_FLOW") {
     var woeid = context.getVariable("request.formparam.woeid");
     context.setVariable("location.woeid", woeid);
}
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
if (context.flow=="TARGET_REQ_FLOW") {
     context.setVariable("request.verb", "GET");
     var woeid = context.getVariable("location.woeid");
     var url = "http://weather.yahooapis.com/forecastrss?"
     context.setVariable("target.url", url + "w=" + woeid);
}
```

1. In SAP API Management UI, open the proxy that you created in the proxy editor.
2. Select the Develop tab.
3. From the New menu, select *New Script*.
4. In the dialog, select JavaScript and give the script a name, like js-example.
5. Paste the following code in the code editor and save the proxy. The important thing to notice is the context object. This object is available to the JavaScript code anywhere in the proxy flow. It's used to obtain flow-specific constants, to call useful get/set methods, and for more operations. This object part is of SAP API Management's JavaScript Object Model. Note, too, that the target.url flow variable is a built-in, read/write variable that is accessible in the Target Request flow. When we set that variable with the "weather" API URL, SAP API Management makes its backend call to that URL. We've essentially rewritten the original target URL, which was whatever you specified when you created the proxy (e.g., http://www.example.com).
6. From the New Policy menu, select JavaScript.
7. Give the policy a name, like target-rewrite. Accept the defaults, and save the policy.
8. If you select the Proxy Endpoint Preflow in the Navigator, you'll see that the policy was added to that flow.
9. In the Navigator, select the Target Endpoint PreFlow icon.
10. From the Navigator, drag the JavaScript policy onto the Request side of the Target Endpoint in the flow editor.
11. Save.
12. Call the API like this, substituting your correct org name and proxy name as appropriate:

```
curl -i -H 'Content-Type: application/x-www-form-urlencoded' -X POST -d
'woeid=12797282' http://myorg-{env}.<host:port>/js-example
```

One final thing, let's take a look at the XML definition for the JavaScript policy used in this example. The important thing to note is that the<ResourceURL> element is used to speicfy the JavaScript source file to execute. This same pattern is used for any JavaScript source file:jsc://filename.js. If you're JavaScript code requires includes, you can use one or more <IncludeURL> elements to do that, as described later in this reference.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Javascript async="false" continueOnError="false" enabled="true" timeLimit="200"
name="target-rewrite">
    <DisplayName>target-rewrite</DisplayName>
    <Properties/>
    <ResourceURL>jsc://js-example.js</ResourceURL>
</Javascript>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.6.3.1.2 Retrieving Property Value from JavaScript

You can add a *<Property>* element in configuration, then retrieve the element's value with JavaScript at runtime.

Use the element's name attribute to specify the name with which to access the property from JavaScript code. The <Property> element's value (the value between the opening and closing tags) is the literal value that will be received by the JavaScript.

In JavaScript, you retrieve the policy property value by accessing it as a property of the Properties object, as in the following:

- Configure the property. Here, the property value is the variable name response.status.code.

```
<Javascript async="false" continueOnError="false" enabled="true" timeLimit="200"
name="JavascriptURLRewrite">
    <DisplayName>JavascriptURLRewrite</DisplayName>
    <Properties>
        <Property name="source">response.status.code</Property>
</Properties>
<ResourceURL>jsc://JavascriptURLRewrite.js</ResourceURL>
</Javascript>
```

- Retrieve the property with JavaScript. Here, the retrieved value -- a variable name -- is then used by the getVariable function to retrieve the variable's value.

```
var responseCode = properties.source; // Returns "response.status.code"
var value = context.getVariable(responseCode); // Get the value of
response.status.code
context.setVariable("response.header.x-target-response-code", value);
```

8.6.3.2 Element reference

The element reference describes the elements and attributes of the JavaScript policy.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Javascript async="false"
        continueOnError="false" enabled="true" timeLimit="200"
        name="JavaScript-1">
    <DisplayName>JavaScript 1</DisplayName>
    <Properties>
        <Property name="propName">propertyValue<Property>
    </Properties>
    <ResourceURL>jsc://my-javascript-source-file</ResourceURL>
    <IncludeURL>jsc://a-javascript-library-file</IncludeURL>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
</Javascript>
```

8.6.3.3 <Javascript> Attributes

```
<Javascript name="Javascript-1" enabled="true" continueOnError="false" async="false"
timeLimit="200">
```

The following attributes are specific to this policy.

Attribute	Description	Default	Presence
name	The internal name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the <DisplayName> element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Required
continueOnError	Set to false to return an error when a policy fails. This is expected behavior for most policies. Set to true to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to true to enforce the policy. Set to false to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to true, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to true improves API proxy performance. However, overusing async can hurt performance with too much thread switching. To use asynchronous behavior in API proxies, see JavaScript callouts.	false	Optional

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.6.3.4 <DisplayName> element

Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

```
<DisplayName>Policy Display Name</DisplayName>
```

Default:	N/A If you omit this element, the the value of the policy'sname attribute is used.
Presence:	Optional
Type:	String

8.6.3.5 <IncludeURL> element

Specifies a JavaScript library file to be loaded as dependency to the main JavaScript file specified with the <ResourceURL> element. The scripts will be evaluated in the order in which they are listed in the policy. Your code can use the objects, methods, and properties of the JavaScript object model.

Include more than one JavaScript dependency resource with additional <IncludeURL> elements.

> **Note**
>
> If your JavaScript files are stored at the organization or environment level, be sure they were uploaded correctly with cURL using the -Foption or as a file attachment through a REST client. Content-Type is `multipart/form-data`.

```
<IncludeURL>jsc://my-javascript-dependency.js</IncludeURL>
```

Default:	None
Presence:	Optional
Type:	String

8.6.3.6 <Property> element

Specifies a property you can access from JavaScript code at runtime.

```
<Properties>
  <Property name="propName">propertyValue</Property>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

</Properties>

Default: None

Presence: Optional

Type: String

Attributes

Attribute	Description	Default	Presence
name	Specifies the name of the property.	N/A	Required.

8.6.3.7 <ResourceURL> element

Specifies the main JavaScript file that will execute in the API flow. You can store this file at the API proxy scope (under /apiproxy/resources/jsc in the API proxy bundle or in the Scripts section of the API proxy editor's Navigator pane), or at the organization or environment scopes for reuse across multiple API proxies, as described in Resource files. Your code can use the objects, methods, and properties of the JavaScript object model.

```
<ResourceURL>jsc://my-javascript.js</ResourceURL>
```

Default: None

Presence: Required

Type: String

8.6.3.8 Flow Variables

This policy does not populate any variables by default; however, you can set (and get) flow variables in your JavaScript code by calling methods on the context object. A typical pattern looks like this:

context.setVariable("response.header.X-SAP-Target", context.getVariable("target.name"))

The context object is part of the SAP API Management JavaScript Object Model.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.6.3.9 Error codes

This policy defines the following error codes. For guidance on handling errors, see Fault handling.

The default format for error codes returned by policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

The Javascript Policy type defines the following error codes:

Error Code	Message
ScriptExecutionFailed	Execution of {0} failed with error: {1}
ScriptExecutionFailedLineNumber	Execution of {0} failed on line {2} with error: {1}
ScriptCompilationFailed	Compilation of JavaScript {0} failed with error:{1}. Context {2}
NoResourceForURL	Could not locate a resource with URL {0}
WrongResourceType	Resource {0} is the wrong type. It is {1}: but Javascript steps use type jsc:

8.6.4 Customize an API Using Python

SAP API Management provides a range of out-of-the-box policy capabilities that address common API management requirements. However, there are some cases where your API requires custom behavior that is not covered by SAP API Management's standard policy palette. In these cases, SAP API Management's exposes scripting interfaces to ease the task of implementing custom behaviors in the proxied API message flow. One approach is to attach your own Python script to an API flow, which the API Platform then executes at runtime.

A Python policy contains no actual code. Instead, a Python policy references a Python 'resource' and defines the Step in the API flow where the Python script executes. You can upload your script through the Management UI proxy editor, or you can include it in the /resources/py directory in API proxies that you develop locally.

> **Note**
>
> System calls, for example network I/O, filesystem read/writes, current user info, process list, and CPU/memory utilization are not permitted by the security model. Although some such calls may be

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

functional, they are unsupported and liable to be actively disabled at any time. For forward compatibility, you should avoid making such calls in your Python scripts.

8.6.4.1 Configuring a Python Script Policy

Configure the Python Script policy using the following elements.

Note

The name attribute for this policy is restricted to these characters: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.

Field Name	Description
ResourceURL	Specifies the name of the Python script stored in the API proxy under /resources/py. Note: the filename must match exactly or an InternalClassification error will be thrown at runtime.
IncludeURL (Optional)	You can include zero or more of these elements. Each elemnt should specify a single Python script in the same form as the ResourceURL element. Scripts are evaluated in the order in which they appear in the policy.

Example

Python Script policy

In the example below, the element, ResourceURL specifies the relevant Python script resource.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Script name="Python-1">
        <DisplayName>Python-1</DisplayName>
        <FaultRules/>
        <ResourceURL>py://myscript.py</ResourceURL>
</Script>
```

Example

Python Script

This shows what you might include in the python script itself.

```
import base64

username = flow.getVariable("request.formparam.client_id")
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

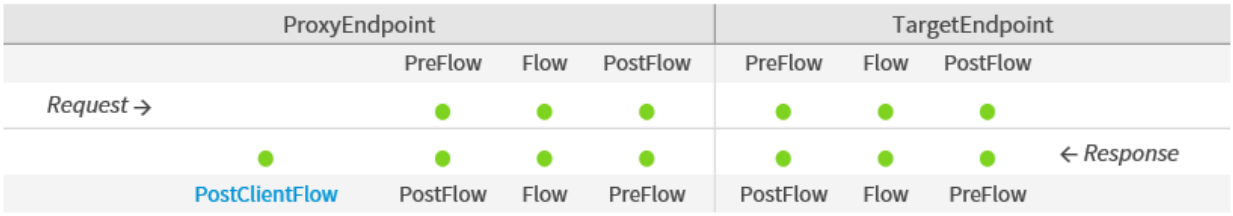
```
password = flow.getVariable("request.formparam.client_secret")

base64string = base64.encodestring('%s:%s' % (username, password))[:-1]
authorization = "Basic "+base64string

flow.setVariable("authorizationParam",authorization)
```

8.6.5 Message Logging Policy

One of the best ways to track down problems in the API runtime environment is to log messages.

SAP API Management enables you to attach and configure a Message Logging policy on your API to log custom messages to a local disk or to syslog.

The policy can be attached in the locations mentioned in this image:



8.6.5.1 Samples

The following samples illustrate different message logging configurations.

Syslog

A common usage of the MessageLogging policy type is to log to a syslog account. When configured for syslog, an API proxy will forward log messages from SAP API Management to a remote syslog server. You may already have a sylog server available. If not, public log management services, such a loggly, are available.

For example, imagine that you need to log information about each request message that your API receives from consumer apps. The value 3f509b58 represents a key value specific to the loggly service. If you have a loggly account, substitute your loggly key. The log message that is generated will be populated with four values: the organization, API proxy, and environment name associated with the transaction, along with the value for a query parameter on the request message.

```
<MessageLogging name="LogToSyslog">
  <Syslog>
    <Message>[3f509b58 tag="{organization.name}.{apiproxy.name}.{environment.name}"]
Weather request for WOEID {request.queryparam.w}.</Message>
    <Host>logs-01.loggly.com</Host>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <Port>514</Port>
    <Protocol>TCP</Protocol>
    <FormatMessage>true</FormatMessage>
  </Syslog>
  <logLevel>ALERT</logLevel>
</MessageLogging>
```

Note:

Host and Port elements cannot refer to variables. They must contain static values.

Log Destination

A number of configurations are supported for logging messages to a file on the SAP API Management management server.

Syslog over TLS/SSL

```
<MessageLogging name="LogToSyslog">
  <Syslog>
    <Message>[3f509b58 tag="{organization.name}.{apiproxy.name}.{environment.name}"]
Weather request for WOEID {request.queryparam.w}.</Message>
    <Host>logs-01.loggly.com</Host>
    <Port>6514</Port>
    <Protocol>TCP</Protocol>
    <FormatMessage>true</FormatMessage>
    <SSLInfo>
        <Enabled>true</Enabled>
    </SSLInfo>
  </Syslog>
  <logLevel>WARN</logLevel>
</MessageLogging>
```

You can send messages to third-party message logging providers over TLS/SSL by adding the <SSLInfo> block.

File rotation based on file size

```
<MessageLogging name="LogPolicy">
  <File>
    <Message>This is a test message. Message id : {request.header.messageId}</Message>
      <FileName>test.log</FileName>
      <FileRotationOptions rotateFileOnStartup="true">
        <FileRotationType>SIZE</FileRotationType>
        <MaxFileSizeInMB>10</MaxFileSizeInMB>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        <MaxFilesToRetain>10</MaxFilesToRetain>
      </FileRotationOptions>
  </File>
</MessageLogging>
```

File rotation based on time

```
<MessageLogging name="LogPolicy">
  <File>
    <Message>This is a test message. Message id : {request.header.messageId}</Message>
    <FileName>test.log</FileName>
    <FileRotationOptions rotateFileOnStartup="true">
      <FileRotationType>TIME</FileRotationType>
      <RotationFrequency unit="minute">10</RotationFrequency>
      <MaxFilesToRetain>10</MaxFilesToRetain>
    </FileRotationOptions>
  </File>
</MessageLogging>
```

File rotation based on time and size

```
<MessageLogging name="LogPolicy">
  <File>
    <Message>This is a test message. Message id : {request.header.messageId}</Message>
    <FileName>test.log</FileName>
    <FileRotationOptions rotateFileOnStartup="true">
      <FileRotationType>TIME_SIZE</FileRotationType>
      <MaxFileSizeInMB>10</MaxFileSizeInMB>
      <MaxFilesToRetain>10</MaxFilesToRetain>
      <RotationFrequency unit="minute">10</RotationFrequency>
    </FileRotationOptions>
  </File>
</MessageLogging>
```

Stream-enabled message logging

```
<MessageLogging name="LogPolicy">
  <File>
  ....
  ....
  </File>
  <BufferMessage>true</BufferMessage>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
</MessageLogging>
```

If HTTP streaming is enabled in an API proxy, neither request nor response messages will be buffered by processing pipeline. When you need to log parsed message content, set BufferMessage to `true`.

8.6.5.2 Element reference

Use the following elements to configure the Message Logging policy type.

> **Note**
>
> The `name` attribute for this policy is restricted to these characters: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.

Field Name		Description
`File` Local file destination. (File logging is only supported in on-premises deployments of SAP API Management.)	Mesage	Build the message to be sent to the log file, combining text with variables to capture the information you want.
	FileName	Name of the log file where the message is logged.
	rotateFileOnStartup	Valid values: true/false If set to true, then the log file is rotated every time the messaging engine restarts.
	FileRotationType	Specifies the rotation policy (size or time) of a log file.
	MaxFileSizeInMB	(On selecting size as rotation type) Specifies the size of a log file that triggers the server to move log messages to a separate file. After the log file reaches the specified size, the server renames the current log file.
	RotationFrequency	(On selecting time as rotation type) Specifies the time in minutes that triggers the server to move log messages to a separate file. After the specified interval elapses, the current log file is renamed.
	MaxFilesToRetain	Specifies the maximum number of files to be retained on rotation policy.
	BufferMessage	If HTTP streaming is enabled for your proxy, request/response messages are not buffered. If you want to log content that requires the flow message to be parsed, then

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

		set BufferMessage to true. See the "Stream-enabled" sample tab for an example. Default: false
`Syslog` destination To send syslog to Splunk, Sumo Logic, or Loggly, see Configuring third-party log management services topic.	Message	Build the message to be sent to the syslog, combining text with variables to capture the information you want. Note: Response variables will not be available in PostClientFlow following an Error Flow. Use message variables to log response information for both error and success situations. See the Usage section below.
	Host	IP address of the syslog server
	Port	Port where the syslog is running. If you don't include this element, the default is 514.
	Protocol	TCP or UDP (default). While UDP is more performant, the TCP protocol guarantees message log delivery to the syslog server. For sending syslog messages over TLS/SSL, only TCP is supported.

	FormatMessage	`true or false`
		Optional, but `<FormatMessage>true</FormatMessage>` is required for use with Loggly.
		This element lets you control the format of SAP API Management-generated content prepended to the message. If set to true, the syslog message is prepended by a fixed number of characters, which lets you filter out that information from messages. Here's an example for the fixed format:
		`<14>1 2016-02-23T09:24:39.039+0000 e49cd3a9-4cf6-48a7-abb9-7ftfe4d97d00 SAP API Management-Edge - - - Message starts here`
		The SAP API Management-generated information includes:
		· <14> - A priority score (see the Syslog Protocol) based on the log level and facility level of the message.
		· 1 - The current syslog version.
		· Date with a UTC offset (UTC = +0000).
		· Message processor UUID.
		· "SAP API Management Edge - - - "

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

	SSLInfo	Lets you log messages through SSL/TLS. Use with sub-element `<Enabled>true</Enabled>`.
		Only the TCP protocol is supported.
		If you don't include this element or leave it empty, the default value is false (no TLS/SSL).
		`<SSLInfo>`
		`<Enabled>true</Enabled>`
		`</SSLInfo>`
		
		Caution:
		TLS/SSL on the syslog server must be configured with a valid CA certificate. Self-signed certificates are not currently supported.

`logLevel`	Optional. Valid values: `INFO` (default), `ALERT`, `WARN`, `ERROR` Set a specific level of information to be included in the message log. If you're using the `FormatMessage` element (setting it to true), your `logLevel` setting affects the calculated priority score (the number inside the angle brackets) in the SAP API Management-generated information prepended to the message.	

8.6.5.3 Usage Notes

When attaching a Message Logging policy to an API proxy flow, consider placing it in the ProxyEndpoint response, in a special flow called PostClientFlow. The PostClientFlow executes after the response is sent to the requesting client, which ensures that all metrics are available for logging.

Note: It's important to note that the response flow variable is not available in the PostClientFlow once processing enters the error state. Instead, you can use the message variable to set values in the MessageLogging policy in the PostClientFlow, ensuring that they will be set whether or not the error state was the previous context.

The PostClientFlow is special in two ways:

1.It only executed as part of the response flow.

2.It is the only flow executed after the proxy enters the error state.

Because it is executed regardless of whether the proxy succeeded or failed, you can put Message Logging policies in the PostClientFlow and be guaranteed that they always execute.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Shown below is the ProxyEndpoint definition that includes the PostClientFlow:

```
<ProxyEndpoint name="default">
  ...
  <PostClientFlow>
    <Response>
      <Step>
        <Name>Message-Logging-1</Name>
      </Step>
    </Response>
  </PostClientFlow>
  ...
</ProxyEndpoint>
```

SAP API Management Edge logs messages as simple text, and you can configure logging to include variables, such as the date and time when the request or response was received, the user identity on the request, the source IP address from which the request was sent, and so on. SAP API Management Edge logs message asynchronously, meaning that no latency that might be caused by blocking callouts is introduced to your API.

The MessageLogging policy writes logged messages in memory to a buffer. The message logger reads messages from the buffer and then writes to the destination that you configure. Each destination has its own buffer.

If the write rate to the buffer increases beyond the read rate, the buffer overflows and logging will fail. If you encounter this issue in an on-premises deployment of SAP API Management Edge, locate the message-logging.properties and use this solution:

Increase the `max.log.message.size.in.kb` property (default value = 128 KB) in the `message-logging.properties` file.

Note:

The response message variables in SAP API Management Edge are not available from the Error Flow. These variables are also not available in PostClientFlow if the preceding flow was the Error Flow. If you want to log response information from the PostClientFlow, use the message object. You can use this object to get at headers and other information from the response whether or not there was an error.

8.6.5.4 Log file location in SAP API Management Edge

By default, message logs are located in the following location on message processors:

`/opt/apigee/var/log/edge-message-processor/messagelog/{org}/{environment}/{api_proxy_name}/{revision}/{logging_policy_name}/`

You can change the default log location by modifying the following properties in the message-logging.properties file on the message processors:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- data.dir - Sets the root path for log file storage. For example, `data.dir=/opt/apigee/var/log`
- log.root.dir - If you set this to a relative path, such as `log.root.dir=custom/folder/`, the path is appended to the data.dir location.

  For example, the combination of the two properties would set the logging directory at`/opt/apigee/var/log/custom/folder/messagelog/` (note that /messagelog is added automatically).

  If you set this to an absolute path, such as `log.root.dir=/opt/apigee/var/log/messages`, message logs will be stored in `/opt/apigee/var/log/messages/messagelog/`. An absolute path in`log.root.dir` takes precedence over data.dir.

If you want to store log files in a flat file structure so that all log files are put in the same directory, `set the enable.flat.directory.structure` property to true in the `message-logging.properties` file on message processors. Messages are stored in the directory specified by the properties above, and the file names take the form of `{org}_{environment}_{api_proxy_name}_{revision}_{logging_policy_name}_{filename}`.

8.6.5.5 Default values for variables in message template

Default values can be specified for each variable in message template separately. For example, if the `variablerequest.header.id` cannot be resolved, then its value is replaced with the value unknown.

```
<Message>This is a test message. id = {request.header.id:unknown}</Message>
```

A common default value can be specified for all the unresolved variables by setting the `defaultVariableValue` attribute on the the Message element:

```
<Message defaultVariableValue="unknown">This is a test message. id =
{request.header.id}</Message>
```

Note:

The default variable value cannot contain spaces.

8.6.5.6 Configuring third-party log management services

The Message Logging policy lets you send syslog messages to third-party log management services, such as Splunk, Sumo Logic, and Loggly. If you want to send syslog to one of those services, see that service's documentation to configure the service's host, port, and protocol, then set the Syslog element on this policy accordingly.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.6.5.7 Flow variables

The following variables are populated on policy failure.

- `messagelogging.failed`
- `messagelogging.{stepdefinition-name}.failed`

Note:

Proxy calls succeed when logging fails

There's a difference between policy errors and message logging errors. The flow variables here are populated only when the policy itself fails, not when message logging fails. Because message logging is first written to buffer, the API proxy will continue successful execution even if message logging ultimately fails (for example, if there's a connection failure to the external syslog provider). Be sure to check your logs on a regular basis to make sure logging is happening as expected

8.7 Mediation Policies

8.7.1 ExtractVariables Policy

Use the ExtractVariables policy to assign variables to URI paths, query parameters, HTTP headers, form parameters, and business-specific information in request or response message content in XML or JSON format. Other policies and code can then reference these variables to enable dynamic behavior.

> **Note**
>
> SAP API Management sets numerous variables automatically during request processing. See Variables reference. Use the ExtractVariables policy to set additional variables, with values derived from message content.

ExtractVariables can also be used to build content-driven Analytics reports. See Analyze API message content using custom analytics.)

Need a more detailed overview of this policy? See Extract message content using ExtractVariables

Need policy error codes? See Error Codes for the ExtractVariables policy.

8.7.1.1 Extract Message Content Using ExtractVariables

Extract content from the request or response messages, including headers, URI paths, JSON / XML payloads, form parameters, and query parameters. The policy works by applying a text pattern to the message content and, upon finding a match, sets a variable with the specified message content.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

After extracting the specified message content to the variable, you can reference the variable in other policies as part of processing a request and response.

This policy can be attached in the following locations.

8.7.1.2 Samples

These policy code samples illustrate how to extract variables from the following types of artifacts:

8.7.1.2.1 URIs

```
<ExtractVariables name="ExtractVariables-1">
   <DisplayName>Extract a portion of the url path</DisplayName>
   <Source>request</Source>
   <URIPath>
      <Pattern ignoreCase="true">/accounts/{id}</Pattern>
   </URIPath>
   <VariablePrefix>urirequest</VariablePrefix>
   <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</ExtractVariables>
```

Consider the sample policy code above. The <URIPath> element tells the Extract Variables policy to extract information from the URI path. The<Pattern> element specifies the pattern to apply to the URI path. The pattern is treated as a simple template, with the curly braces denoting the varying portion of the URI path.

The name of the variable to be set is determined by the value specified in the <VariablePrefix> element, as well as the value enclosed in curly braces {} in the <Pattern> element. The two values are joined by an intervening dot, resulting in a variable name of urirequest.id for example. If there is no <VariablePrefix> element, then the variable name is just the value enclosed in curly braces.

Consider the sample policy code above working with the following incoming request:

```
GET http://org1-{env}.<host:port>/accounts/12797282
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

When SAP API Management applies the Extract Variables policy code above to this incoming request, it sets the variable urirequest.id to 12797282. After SAP API Management executes the policy, subsequent policies or code in the processing flow can refer to the variable named urirequest.id to get the string value 12797282.

You can now access the variable urirequest.id in your proxy. For example, the following AssignMessage policy copies it to the payload of the request:

```
<AssignMessage async="false" continueOnError="false" enabled="true" name="GetURIPath">
 <DisplayName>GetURIPath</DisplayName>
  <Set>
   <Payload contentType="text/xml">
    <ExtractURI>{urirequest.id}</ExtractURI>
   </Payload>
  </Set>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
  <AssignTo createNew="false" transport="http" type="request"/>
</AssignMessage>
```

8.7.1.2.2 Query Params

```
<ExtractVariables name="ExtractVariables-2">
   <DisplayName>Extract a value from a query parameter</DisplayName>
   <Source>request</Source>
   <QueryParam name="code">
      <Pattern ignoreCase="true">DBN{dbncode}</Pattern>
   </QueryParam>
   <VariablePrefix>queryinfo</VariablePrefix>
   <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</ExtractVariables>
```

Consider the sample policy code above. Suppose that your API design stipulates that incoming requests must carry a query parameter named code.code holds a term that looks like DBNXXXXX, where DBN is fixed and the XXXXX denotes a varying string. You can use this policy to extract the varying string.

Consider the sample policy code above working with the following incoming request:

```
GET http://org1-{env}.<host:port>/accounts/12797282?code=DBN88271
```

When SAP API Management applies the Extract Variables policy code above to this incoming request, it sets the variable queryinfo.dbncode to 88271. After SAP API Management executes the policy, subsequent policies or code in the processing flow can refer to the variable named queryinfo.dbncode to get the string value 88271.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

You can now access the variable queryinfo.code in your proxy. For example, the following AssignMessage policy copies it to the payload of the request:

```
<AssignMessage async="false" continueOnError="false" enabled="true" name="GetURIPath">
 <DisplayName>GetQP</DisplayName>
  <Set>
   <Payload contentType="text/xml">
    <ExtractQP>{queryinfo.code}</ExtractQP>
   </Payload>
  </Set>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
  <AssignTo createNew="false" transport="http" type="request"/>
</AssignMessage>
```

8.7.1.2.3 Multiple Params

```
<ExtractVariables name="ExtractVariables-2">
   <DisplayName>Extract a value from a query parameter</DisplayName>
   <Source>request</Source>
   <QueryParam name="w">
      <Pattern ignoreCase="true">{firstWeather}</Pattern>
   </QueryParam>
   <QueryParam name="w.2">
     <Pattern ignoreCase="true">{secondWeather}</Pattern>
   </QueryParam>
   <VariablePrefix>queryinfo</VariablePrefix>
 <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</ExtractVariables>
```

Suppose your API design allows you to specify multiple query paramaters with the same name. You can use this policy to extract the value of multiple instances of the query parameter "w". To reference these query parameters in the Extract Variables policy, you use indexes, where the first instance of the query parameter has no index, the second is at index 2, the third at index 3, etc.

Consider the sample policy code above working with the following incoming request:

```
GET http://org1-{env}.<host:port>/weather?w=Boston&w=Chicago
```

When SAP API Management applies the Extract Variables policy code above to this incoming request, it sets the variable queryinfo.firstWeather toBoston and the variable queryInfo.secondWeather to Chicago.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

You can now access the variable queryinfo.firstWeather and queryinfo.secondWeather in your proxy. For example, the following AssignMessage policy copies it to the payload of the request:

```
<AssignMessage async="false" continueOnError="false" enabled="true" name="GetURIPath">
 <DisplayName>GetQP</DisplayName>
  <Set>
   <Payload contentType="text/xml">
    <ExtractQP1>{queryinfo.firstWeather}</ExtractQP1>
    <ExtractQP2>{queryinfo.secondWeather}</ExtractQP2>
   </Payload>
  </Set>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
  <AssignTo createNew="false" transport="http" type="request"/>
</AssignMessage>
```

8.7.1.2.4 Headers

```
<ExtractVariables name='ExtractVariable-OauthToken'>
  <Source>request</Source>
  <Header name="Authorization">
    <Pattern ignoreCase="false">Bearer {oauthtoken}</Pattern>
  </Header>
  <VariablePrefix>clientrequest</VariablePrefix>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</ExtractVariables>
```

Suppose that your API uses OAuth v2.0 bearer tokens. Consider the sample policy code above working with a request carrying an OAuth v2.0 token that includes a header like this: Authorization: Bearer TU08xptfFfeM7aS0xHqlxTgEAdAM.

As the API designer, suppose that you want to use the token value (but not the entire header) as a key in a cache lookup. You could use the Extract Variables policy code above to extract the token.

When SAP API Management applies the Extract Variables policy code above to this header, it will set the variable clientrequest.oauthtoken toTU08xptfFfeM7aS0xHqlxTgEAdAM.

You can now access the variable clientrequest.oauthtoken in your proxy. For example, the following AssignMessage policy copies it to the payload of the request:

```
<AssignMessage async="false" continueOnError="false" enabled="true" name="GetURIPath">
 <DisplayName>GetHeader</DisplayName>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  <Set>
   <Payload contentType="text/xml">
    <ExtractHeader>{clientrequest.oauthtoken}</ExtractHeader>
   </Payload>
  </Set>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
  <AssignTo createNew="false" transport="http" type="request"/>
</AssignMessage>
```

8.7.1.2.5 JSON

```
<ExtractVariables name="ExtractVariables-3">
   <Source>response</Source>
   <JSONPayload>
      <Variable name="latitude" type="float">
         <JSONPath>$.results[0].geometry.location.lat</JSONPath>
      </Variable>
      <Variable name="longitude" type="float">
         <JSONPath>$.results[0].geometry.location.lng</JSONPath>
      </Variable>
   </JSONPayload>
   <VariablePrefix>geocoderesponse</VariablePrefix>
</ExtractVariables>
```

The Extract Variables policy can extract values from complex structures, such as JSON messages. The sample policy code above shows how to extract a variable from a portion of a JSON message payload.
The <JSONPayload> element tells the policy to extract a variable from a JSON payload. You specify the portion to extract using a JSON path expression in which the $ character refers to the root node of the JSON message.

Consider the following JSON response payload:

```
{
  "results": [{
    "geometry": {
      "location": {
        "lat": 37.42291810,
        "lng": -122.08542120
      },
      "location_type": "ROOFTOP",
      "viewport": {
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        "northeast": {
          "lat": 37.42426708029149,
          "lng": -122.0840722197085
        },
        "southwest": {
          "lat": 37.42156911970850,
          "lng": -122.0867701802915
        }
      }
    }
  }]
}
```

When SAP API Management applies the Extract Variables policy code above to this JSON message, it sets two variables: geocoderesponse.latitude andgeocoderesponse.longitude. Both variables use the same variable prefix of geocoderesponse. The suffix for these variables is specified explicitly by the <Variable> element's name attribute.

The variable geocoderesponse.latitude gets the value 37.42291810. The variable geocoderesponse.longitude gets the value-122.08542120.

You can now access the variable geocoderesponse.latitude in your proxy. For example, the following AssignMessage policy copies it to a header named "latitude" in the response:

```
<AssignMessage async="false" continueOnError="false" enabled="true" name="GetURIPath">
  <DisplayName>GetJSONVar</DisplayName>
  <Add>
    <Headers>
      <Header name="latitude">{geocoderesponse.latitude}</Header>
    </Headers>
  </Add>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
  <AssignTo createNew="false" transport="http" type="response"/>
</AssignMessage>
```

8.7.1.2.6 XML

```
<ExtractVariables name="ExtractVariables-4">
   <Source>response</Source>
   <XMLPayload>
      <Namespaces>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
            <Namespace prefix="dir">urn:43BFF88D-D204-4427-B6BA-140AF393142F</Namespace>
        </Namespaces>
        <Variable name="travelmode" type="string">
            <XPath>/dir:Directions/dir:route/dir:leg/dir:step/@mode</XPath>
        </Variable>
        <Variable name="duration" type="string">

<XPath>/dir:Directions/dir:route/dir:leg/dir:step/dir:duration/dir:value</XPath>
        </Variable>
        <Variable name="timeunit" type="string">

<XPath>/dir:Directions/dir:route/dir:leg/dir:step/dir:duration/dir:text</XPath>
        </Variable>
    </XMLPayload>
    <VariablePrefix>directionsresponse</VariablePrefix>
</ExtractVariables>
```

The Extract Variables policy can extract values from complex structures, such as XML messages. The sample policy code above shows how to extract a variable from a portion of a XML message payload.
The <XMLPayload> element tells the policy to extract a variable from an XMLpayload. You specify the portion to extract using XPath and explicitly named variables.

Consider the following XML response payload:

```
<Directions xmlns="urn:43BFF88D-D204-4427-B6BA-140AF393142F">
    <status>OK</status>
    <route>
        <summary>I-40 W</summary>
        <leg>
            <step mode="DRIVING">
                <start_location>
                    <lat>41.8507300</lat>
                    <lng>-87.6512600</lng>
                </start_location>
                <end_location>
                    <lat>41.8525800</lat>
                    <lng>-87.6514100</lng>
                </end_location>
                <duration>
                    <value>19</value>
                    <text>minutes</text>
                </duration>
            </step>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
      </leg>
   </route>
</DirectionsResponse>
```

When SAP API Management applies the Extract Variables policy code above to this XML message, it sets three variables:directionsresponse.travelmode, directionsresponse.duration, and directionsresponse.timeunit. All variables use the same variable prefix of directionsresponse. The suffix for these variables is specified explicitly by the <Variable> element's name attribute.

The variable directionsresponse.travelmode gets the value DRIVING. The variable directionsresponse.duration gets the value 19. The variable directionsresponse.timeunit gets the value minutes.

You can now access the variable directionresponse.travelmode in your proxy. For example, the following AssignMessage policy copies it to a header named "tmode" in the response:

```
<AssignMessage async="false" continueOnError="false" enabled="true" name="GetURIPath">
  <DisplayName>GetXMLVar</DisplayName>
  <Add>
    <Headers>
      <Header name="tmode">{directionsresponse.travelmode}</Header>
    </Headers>
  </Add>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
  <AssignTo createNew="false" transport="http" type="request"/>
</AssignMessage>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.1.3 Overview of the ExtractVariables policy

API Developers build policies that behave differently based on the content of messages, including headers, URI paths, payloads, and query parameters.

Often, API Developers will want to extract some portion of a message, or of a header, so that they can use that extracted portion in a condition statement. The ExtractVariables policy does this. It applies a text pattern to some part of the message - the body or header or URL path or a query parameter - and on finding a match, sets a designated variable with the appropriate content.

Other policies and code can then consume those variables to enable dynamic behavior or to send business data to Analytics Services.

To see how ExtractVariables can be used to build content-driven Analytics reports, see Analyze API message content using custom analytics.

> **Note**
>
> SAP API Management sets numerous variables automatically during request processing. See Variables reference. Use the ExtractVariables policy to set additional variables, with values derived from message content.

8.7.1.4 About matching and variable creation

The Extract Variables policy extracts information from a request or response and writes that information to a variable. For each type of information that you can extract, such as URI path or XML data, you specify the pattern to match and the name of the variable used to hold the extracted information.

However, the way pattern matching works depends on the source of the extraction. The following sections describe the two basic categories of information that you can extract.

8.7.1.4.1 Matching URI paths, query parameters, headers, form parameters, and variables

When extracting information from a URI path, query parameters, headers, form parameters, and variables you use the <Pattern> tag to specify one or more patterns to match. For example, the following policy example shows a single matching pattern for the URI path:

```
<ExtractVariables name="ExtractVariables-1">
   <Source>request</Source>
   <URIPath>
      <Pattern ignoreCase="true">/a/{pathSeg}</Pattern>
   </URIPath>
   <VariablePrefix>urirequest</VariablePrefix>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
   <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</ExtractVariables>
```

In this example, the urirequest.pathSeg variable is set to whatever appears in the URI path after "/a/". For example, for the following request URL, the variable is set to "b":

```
http://myCo.com/a/b
```

Specifying multiple patterns

- You can specify multiple patters to match, corresponding to <Pattern> tags, where:
- All patterns are tested for match.
- If none of the patterns match, the policy does nothing and the variable(s) is not created.
- If more than one pattern matches, the pattern with longest path segments is used for extraction.
- If two matched patterns has same longest path segments, then the pattern specified first in the policy is used for extraction.

In the next example, you create a policy that contains three matching patterns for the URI path:

```
<ExtractVariables name="ExtractVariables-1">
   <Source>request</Source>
   <URIPath>
      <Pattern ignoreCase="true">/a/{pathSeg}</Pattern>
      <Pattern ignoreCase="true">/a/b/{pathSeg}</Pattern>
      <Pattern ignoreCase="true">/a/b/c/{pathSeg}</Pattern>
   </URIPath>
   <VariablePrefix>urirequest</VariablePrefix>
   <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</ExtractVariables>
```

The request URL to the API proxy is in the form:

```
http://myCo.com/a/b
```

In this example, the first pattern matches the URI and the urirequest.pathSeg variable is set to "b". If the request URL is:

```
http://myCo.com/a/b/c/d
```

Then the third pattern matches and the urirequest.pathSeg variable is set to "d".

Specifying patterns with multiple variables

You can specify multiple variables in the matching pattern. For example, you specify a matching pattern with two variables:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<ExtractVariables name="ExtractVariables-1">
   <Source>request</Source>
   <URIPath>
      <Pattern ignoreCase="true">/a/{pathSeg}</Pattern>
      <Pattern ignoreCase="true">/a/b/{pathSeg}</Pattern>
      <Pattern ignoreCase="true">/a/{pathSeg1}/c/{pathSeg2}</Pattern>
   </URIPath>
   <VariablePrefix>urirequest</VariablePrefix>
   <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</ExtractVariables>
```

For the request URL:

```
http://myCo.com/a/b/c/d
```

The urirequest.pathSeg1 variable is set to "b" and the urirequest.pathSeg2 variable is set to "d".

Matching multiple instances in the pattern

You can also match patterns when there are multiple instances of an item with the same name. For example, you can make a request that contains multiple query parameters or multiple headers with the same name. The following request contains two query parameters named "w":

```
http://myCo.com/a/b/c/d?w=1&w=2
```

To reference these query parameters in the Extract Variables policy, you use indexes, where the first instance of the query parameter has no index, the second is at index 2, the third at index 3, etc. For example, the following policy extracts the value of the second query parameter named "w" in the request:

```
<ExtractVariables name="ExtractVariables-1">
   <Source>request</Source>
   <QueryParam name="w.2">
      <Pattern ignoreCase="true">{secondW}</Pattern>
   </QueryParam>
   <VariablePrefix>urirequest</VariablePrefix>
   <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</ExtractVariables>
```

The urirequest.secondW variable is set to "2". If the second query parameter is omitted from the request, then the urirequest.secondW variable is empty. Use indexing any time there are multiple items with the same name in the request.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Using special characters in the pattern

When matching URI paths, you can use the "*" and "**" wildcard characters in the pattern, where:

"*" matches any one segments of the path

"**" matches multiple segments of the path

For example, you specify patterns to the <URIPath> element as shown below:

```
<URIPath>
  <Pattern ignoreCase="true">/a/*/{id}</Pattern>
  <Pattern ignoreCase="true">/a/**/{id}</Pattern>
</URIPath>
```

The first pattern matches URIs such as "/a/b/c", "/a/foo/bar", etc. The second pattern matches any number of path segments after "/a/", such as "/a/foo/bar/baz/c", as well as "/a/b/c" and "/a/foo/bar".

When specifying patterns to query parameters, headers, and form parameters, the "*"character specifies to match any number of characters. For example, when matching a header, specify the pattern as:

*;charset={encoding}

This pattern matches the values "text/xml;charset=UTF-16" and "application/xml;charset=ASCII".

If the value passed to the Extract Variables policy contains a special character, such as "{", use the "%" character to escape it. The following example escapes the "{" and "}" characters in the pattern because they are used as literal characters in the value of the query parameter:

```
<QueryParam>
  <Pattern ignoreCase="true">%{user%} {name}</Pattern>
</QueryParam>
```

In this example, the pattern matches the value "{user} Steve" but not the value "user Steve".

Matching JSON and XML

When extracting data from JSON and XML, you specify one or more <Variable> tags in the policy.
The <Variable> tag specifies the name of the destination variable where the extracted information is stored, and the JsonPath (JSON) or XPATH (XML) to the extracted information.

All <Variable> tags in the policy are evaluated, so that you can populate multiple variables from a single policy. If the <Variable> tag does not evaluate to a valid field in the JSON or XML, then the corresponding variable is not created.

The following example shows an Extract Variables policy that populates two variables from the JSON body of a response:

```
<ExtractVariables name="ExtractVariables-3">
   <Source>response</Source>
   <JSONPayload>
      <Variable name="latitude" type="float">
         <JSONPath>$.results[0].geometry.location.lat</JSONPath>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
      </Variable>
      <Variable name="longitude" type="float">
         <JSONPath>$.results[0].geometry.location.lng</JSONPath>
      </Variable>
   </JSONPayload>
   <VariablePrefix>geocoderesponse</VariablePrefix>
</ExtractVariables>
```

Writing to the same variable in multiple places

The policy executes sequentially from the first extraction pattern to the last. If the policy writes a value to the same variable from multiple places, the last write in the policy determines the value of the variable.

For example, you want to extract a token value that can be passed either in a query parameter or in a header, as shown below:

```
<!-- If token only in query param, the query param determines the value.
     If token is found in both the query param and header, header sets value. -->
<QueryParam name="token">
  <Pattern ignoreCase="true">{tokenValue}</Pattern>
</QueryParam>

<!-- Overwrite tokenValue even if it was found in query parameter. -->
<Header name="Token">
  <Pattern ignoreCase="true">{tokenValue}</Pattern>
</Header>
```

Controlling what happens when no match occurs

If the pattern does not match, then the corresponding variable is not created. Therefore, if another policy references the variable, it can cause an error.

One option is to set <IgnoreUnresolvedVariables> to true in a policy that references the variable to configure the policy to treat any unresolvable variable as an empty string (null):

```
<IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.2 Apply XSL Transformations

Extensible stylesheet language transformations (XSLT) is a language for converting documents from one XML format to another XML format. It is often used to integrate applications that support XML, but that require different XML-formats for the same data.

The XSL policy type enables XSLT to execute as a processing step in an API proxy flow. The XSLT is implemented in a stand-alone .xslt file, which is stored in the API proxy under /resources/xsl. The XSL policy merely references the XSL file. See Resource Files for more.

The XSL policy requires two inputs:

- The name of an XSLT stylesheet, which contains a set of transformation rules) stored in the API proxy under/resources/xsl
- The source of the XML to be transformed (typically a request or response message)

8.7.2.1 Sample

A simple XSL policy is:

```
<XSL name="TransformXML">
  <ResourceURL>xsl://{myfile}.xsl</ResourceURL>
  <Source>request</Source>
</XSL>
```

There are many tools and resources on the Internet implementing XSLT. For reference, below are some XSLT examples with sample request messages and transformed messages.

XSLT File

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="text"/>
  <xsl:variable name="newline">
  <xsl:text>
  </xsl:text>
  </xsl:variable>
  <xsl:template match="/">
  <xsl:text>&lt;Life&gt;</xsl:text>
    <xsl:value-of select="$newline"/>
    <xsl:text>Here are the odd-numbered items from the list:</xsl:text>
    <xsl:value-of select="$newline"/>
    <xsl:for-each select="list/listitem">
      <xsl:if test="(position() mod 2) = 1">
        <xsl:number format="1. "/>
        <xsl:value-of select="."/>
        <xsl:value-of select="$newline"/>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
      </xsl:if>
    </xsl:for-each>
  <xsl:text>&lt;/Life&gt;</xsl:text>
</xsl:template>
</xsl:stylesheet>
```

Request Message

```
<?xml version="1.0"?>
<list>
  <title>A few of my favorite albums</title>
  <listitem>A Love Supreme</listitem>
  <listitem>Beat Crazy</listitem>
  <listitem>Here Come the Warm Jets</listitem>
  <listitem>Kind of Blue</listitem>
  <listitem>London Calling</listitem>
  <listitem>Remain in Light</listitem>
  <listitem>The Joshua Tree</listitem>
  <listitem>The Indestructible Beat of Soweto</listitem>
</list>
```

Transformed Message

```
<Life>
Here are the odd-numbered items from the list:
1. A Love Supreme
3. Here Come the Warm Jets
5. London Calling
7. The Joshua Tree
</Life>
```

SAP API Management relies on the Saxon XSLT processor, and supports XSLT 1.0 and 2.0.

8.7.2.2 Configuring an XSL Transformation Policy

Configure an XSL Transformation policy using the following elements.

> **Note**
>
> `<xsl:include>` and `<xsl:import>` are not supported.

Field Name	Description
Name (Mandatory)	Name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the Management UI

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name			Description
			enforces additional restrictions, such as automatically removing characters that are not alphanumeric.
Source (Optional)			Contains the message from which information needs to be extracted. Usually this value is set to request or response, depending on whether the message to be transformed is inbound or outbound. · If source is missing, it is treated as a simple message. For example,<Source>message</Source> · If the source variable cannot be resolved, or resolves to a non-message type, the transformation step fails.
OutputVariable (Optional)			A variable that stores the output of the transformation. The OutputVariable cannot be of Message type, that is, it cannot be 'message', 'request', or 'response'. You should set this element to be a custom variable, and then consume that variable. To replace the message content with the output of the transformation, delete this element.
ResourceURL (Mandatory)			The XSLT file to be used for transforming the message.
Parameters (Optional)	ignoreUnresolvedVariables (Optional)		Ignores any unresolved variable errors in the XSLT script instructions. Valid values: true/false Default value: false
	Parameter (Optional)	name (Mandatory)	Name of a custom parameter. Note that with name you can only use one of the optional parameters listed below.
		ref (Optional)	Specifies the reference that sources the value from a variable.
		value (Optional)	Value of the parameter.

8.7.2.3 Policy-specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

The XSLT policy type defines the following error codes:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Error Code	Message
XSLSourceMessageNotAvailable	{0} message is not available for XSL: {1}
XSLEvaluationFailed	Evaluation of XSL {0} failed with reason: "{1}"
XSLVariableResolutionFailed	Failed to resolve variable {0}
XSLInvalidResourceType	XSL {0}: Resource type must be xsl. Context {1}
XSLEmptyResourceUrl	Resource Url cannot be empty in XSL {0}

8.7.3 Exception Handling With RaiseFault

The API Platform enables you to configure custom exception handling using a policy of type RaiseFault.

The RaiseFault policy is a variation on the AssignMessage policy type. Like an AssignMessage policy, a RaiseFault policy generates a custom message in response to an error condition. You use RaiseFault to define a FaultResponse that is returned to the requesting app when a specific condition arises.

A FaultResponse can consist of HTTP headers, query parameters, and a message payload. These elements can be dynamically populated using variables, enabling you to craft FaultResponses that are tailored to specific failure conditions. Such tailored FaultResponses can be more useful to app developers and app end users than generic error messages or HTTP response codes.

When executed, the RaiseFault policy transfers the message Flow execution to the default ErrorFlow, which in turn returns the designated FaultResponse to the requesting client app. When the message Flow switches to the default ErrorFlow, no further policy processing occurs. All remaining processing Steps are bypassed, and the FaultResponse is returned directly to the requesting app.

8.7.3.1 Sample RaiseFault Policy

In the most common usage, RaiseFault is used to return a custom FaultResponse to the requesting app. For example to return a 404:

```
<RaiseFault name="404">
 <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
 <FaultResponse>
   <Set>
     <StatusCode>404</StatusCode>
     <ReasonPhrase>The resource requested was not found</ReasonPhrase>
   </Set>
 </FaultResponse>
</RaiseFault>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

A more complex example involves returning a custom FaultResponse payload, along with HTTP headers and an HTTP status code. In the following example the FaultResponse is populated with an XML message along with the value of single variable--the HTTP status code received by the API Platform from the backend service.

```
<RaiseFault name="ExceptionHandler">
 <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
 <FaultResponse>
     <Payload contentType="text/xml">
       <root>Please contact support@company.com</root>
     </Payload>
     <StatusCode>{response.status.code}</StatusCode>
     <ReasonPhrase>Server error</ReasonPhrase>
   </Set>
 <FaultResponse>
<RaiseFault>
```

For a list of all variables that are available for dynamically populating FaultResponse messages, see Variables reference

8.7.3.2 Configuring a Raise Fault Policy

Configure the RaiseFault policy using the following elements.

> **Note**
>
> The name attribute for this policy is restricted to these characters: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.

Field Name	Description
IgnoreUnresolvedVariables: Optional	Ignores any unresolved variable error in the Flow. Valid values: true / false Default value: true
FaultResponse: Optional	Defines the response message returned to the requesting client. FaultResponse uses the same settings as the AssignMessage policy type. See Generate or modify messages using AssignMessage.

> **Example**
>
> RaiseFault policy
>
> ```
> <RaiseFault name="abc">
>  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
> ```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<FaultResponse>
  <Copy source="sc.response">  <!-- source is optional.. default is message -->
     <Headers />
  </Copy>
  <Remove>
     <Headers>
        <Header name="h1"/>
     </Headers>
  </Remove>
  <Set>
    <Headers>
      <Header name="h1">hello {user}</Header>
      <Header name="h2"/>bye {user}</Header>
      <Header name="h3.2"/>bye {user}</Header>
    </Headers>
    <Payload contentType="text/xml">
      <root>this is my payload {variable.v1}</root>
    </Payload>
    <StatusCode>{response.status.code} OR 500</StatusCode>
    <ReasonPhrase>Server error</ReasonPhrase>
  </Set>
 </FaultResponse>
</RaiseFault>
```

Policy-specific variables

Flow variables enable the dynamic behavior of policies and Flows at runtime, based on HTTP headers, message content, or Flow context. The following predefined Flow variables are available after a RaiseFault policy executes. For more information about Flow variables, see Variables reference.

Variable	Type	Permission	Description
fault.name	String	Read-Only	Returns the fault name in the error and if not available, an empty string.
fault.type	String	Read-Only	Returns the fault type in the error and if not available, an empty string.
fault.category	String	Read-Only	Returns the fault category in the error and if not available, an empty string.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.4 Generate or Modify Messages Using AssignMessage

The AssignMessage policy type is used to generate or modify a request or response messages during an API proxy Flow. With this policy, you can assign the verb, the payload, the status code, and more - everything pertaining to an HTTP request or response message.

There are many situations in which a request or a response message must be modified during processing by SAP API Management. For example, you might need to strip HTTP headers from a request message before the message is forwarded to the backend service. You might also need to add HTTP headers or insert JSON or XML message content before a response is sent to the consumer app. You can do both operations using the AssignMessage policy type.

In other situations, you need to generate complete request or response messages. For example, you might use a ServiceCallout policy to invoke a remote API from an API proxy. You can use the AssignMessage policy to generate a request message. Then, employ a ServiceCallout policy that references the variable to send that message to a remote service or API.

AssignMessage is so named because the policy requires a message to be 'assigned' to a variable. (See Variables reference for more on variables.) Often, variables hold string or integer values. In the case of AssignMessage, the variable holds a message - which is a complex data type with multiple fields.

To use AssignMessage, you must select a variable name and specify the message content to assign to it. If you use the one of the built-in names - request or response - the message will be assigned to the ambient request or response, respectively. If you use any other name, it will refer to a custom variable to hold the message.

8.7.4.1 Samples

8.7.4.1.1 Custom Request

This sample shows how to build up a custom request object with Assign Message.

```
<AssignMessage name="AssignMessage-3">
  <AssignTo createNew="true" type="request">request1</AssignTo>
  <Copy>
    <Headers>
     <Header name="user-agent"/>
    </Headers>
  </Copy>
  <Set>
    <QueryParams>
      <QueryParam name="address">{request.queryparam.address}</QueryParam>
    </QueryParams>
    <Verb>GET</Verb>
  </Set>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
</AssignMessage>
```

It's a common use case to build up a custom request object using Assign Message, and then send it to a backend target using the Service Callout policy. This sample:

- Creates a new request message object called request1.
- Copies the value of the `user-agent` HTTP header from the incoming request message object. Because the <Copy> element uses an absolute reference to the user-agent flow variable, there is no need to specify the source attribute to <Copy>.
- Sets a query parameter on the `request1` message object.
- Sets the HTTP verb for the `request1` message object to `GET`.
- Sets the `IgnoreUnresolvedVariables` element to false. So, if one of the variables we are trying to copy or set does not exist, processing in the API flow will stop.

You can then access the request1 object in another Assign Message policy as shown below:

```
<AssignMessage name="CopyExample1ToRequest">
  <DisplayName>AccessRequest1</DisplayName>
  <AssignTo createNew="false" type="request"></AssignTo>
  <Set>
    <Headers>
      <Header name="user-agentCopyRequest1">{request1.header.user-agent}</Header>
    </Headers>
  </Set>
</AssignMessage>
```

8.7.4.1.2 Create Request

Here's another example demonstrating how to create a custom request object with Assign Message.

```
<AssignMessage name="AssignMessage-2">
  <AssignTo createNew="true" type="request">partner.request</AssignTo>
  <Set>
    <Verb>POST</Verb>
    <Payload contentType="text/xml">
      <request><operation>105</operation></request>
    </Payload>
  </Set>
</AssignMessage>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

To work with message content, Assign Message defines a set of elements that enable you to populate or modify HTTP headers, query parameters, and XML or JSON payload content. This sample policy creates a new HTTP request message, and assigns that message to the variable calledpartner.request.

If the createNew attribute is set to false, then the policy will modify the designated message.

8.7.4.1.3 Modify Response

This sample shows how to modify an existing response object by adding a Header to it.

```
<AssignMessage name="assignMessage-4">
  <AssignTo createNew="false" type="response"></AssignTo>
  <Set>
    <Headers>
      <Header name="Cache-Hit">{lookupcache.LookupCache-1.cachehit}</Header>
    </Headers>
  </Set>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
</AssignMessage>
```

This sample Assign Message policy does not create a new message. Instead, it modifies an existing response message by adding an HTTP header. THis policy would be attached to a response flow in the API proxy. Because you omitted a variable name in the <AssignTo> tag, and specified a typeas response, this policy modifies the response object returned by the target server.

The HTTP header added to the response message by this policy is derived from a variable populated by the LookupCache policy. Therefore the response message modified by this AssignMessage policy contains an HTTP header that indicates whether the results have been pulled from the Cache or not. Setting headers in the response can be handy for debugging and troubleshooting.

This pattern of setting headers for debugging and troubleshooting is a common practice. You can see what header value was sent back by inspecting the response with the Trace tool or other means.

8.7.4.1.4 Set Dynamic Content

You can use Assign Message to embed dynamic content in the payload of XML or JSON response and request messages.

To embed SAP API Management flow variables in an XML payload, wrap the designated variable in curly braces, like this: {prefix.name}. This sample policy embeds the value of the HTTP user-agent header flow variable in an XML element called User-agent:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<AssignMessage name="api-token-cache-hit-header">
  <AssignTo createNew="false" type="response"></AssignTo>
  <Set>
    <Payload contentType="text/xml">
      <User-agent>{request.header.user-agent}</User-agent>
    </Payload>
  </Set>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
</AssignMessage>
```

Note: With a JSON payload, you cannot use curly braces to identify variables. Instead, you must declare and embed the delimiter characters, as follows:

```
<Payload contentType="application/json" variablePrefix="@" variableSuffix="#">
{
   "user-agent": "@request.header.user-agent#"
}
</Payload>
```

8.7.4.1.5 Remove apikey

This policy removes the apikey query parameter from the request.

```
<AssignMessage name="AssignMessage-1">
  <AssignTo createNew="false" type=request></AssignTo>
  <Remove>
    <QueryParams>
      <QueryParam name='apikey'/>
    </QueryParams>
  </Remove>
</AssignMessage>
```

It's a best practice to strip the apikey query parameter from the request message when you use the APIKey policy for user authentication. You do this to prevent sensitive key information from being passed to the backend target. This sample pattern is a common use case for AssignMessage.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.4.2 About the Assign Message policy

Use the Assign Message policy to generate or modify request or response messages during an API proxy Flow. With this policy, you can assign the verb, the payload, the status code, and more - everything pertaining to an HTTP request or response message.

Assign Message is so named because the policy requires a message to be 'assigned' to a variable. In the case of Assign Message, the variable holds a message - which is a complex data type with multiple fields.

To use Assign Message, you must select a variable name and specify the message content to assign to it. If you use the one of the built-in names -request or response - the message will be assigned to the request or response, respectively. If you use any other name, it refers to a custom variable that holds the message.

There are many situations in which a request or a response message must be modified during processing by SAP API Management. For example, you might need to strip HTTP headers from a request message before the message is forwarded to the backend service. You might also need to add HTTP headers or insert JSON or XML message content before a response is sent to the consumer app. You can do both operations using the Assign Message policy type.

In other situations, you need to generate complete request or response messages. For example, you might use a ServiceCallout policy to invoke a remote API from an API proxy. You can use the AssignMessage policy to generate the request message and assign it to a variable. Then, use a ServiceCallout policy that references the variable to send that message to the remote API.

8.7.4.3 Element reference

The element reference describes the elements and attributes of the Assign Message policy.

> **Note**
>
> This policy handles a number of use cases. Typically, you select which operation(s) you wish to perform with this policy and delete the rest of the elements. For example, if you know you want to do a Copy, then you can delete the Remove, Add, and other elements if you wish.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AssignMessage async="false" continueOnError="false" enabled="true" name="a_name">
    <DisplayName>a_display_name</DisplayName>
    <AssignTo createNew="false" transport="http" type="request"/>
    <Copy source="request">
        <Headers/>
        <QueryParams/>
        <FormParams/>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        <Payload/>
        <Verb/>
        <Version/>
        <StatusCode/>
        <ReasonPhrase/>
        <Path/>
    </Copy>
    <Remove>
        <Headers>
            <Header name="a_header_name"></Header>
        </Headers>
        <QueryParams>
            <QueryParam name="a_query_param_name"></QueryParam>
        </QueryParams>
        <FormParams>
            <FormParam name="a_form_param_name"></FormParam>
        </FormParams>
        <Payload> </Payload>
    </Remove>
    <Add>
        <Headers/>
        <QueryParams/>
        <FormParams/>
    </Add>
    <Set>
        <Headers/>
        <QueryParams/>
        <FormParams/>
        <Payload/>
        <Verb/>
        <Version/>
        <Path/>
        <ReasonPhrase/>
        <StatusCode/>
    </Set>
    <AssignVariable>
        <Name>name</Name>
        <Value/>
        <Ref/>
    </AssignVariable>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
</AssignMessage>
```

8.7.4.3.1 <AssignMessage> attributes

```
<AssignMessage async="false" continueOnError="false" enabled="true" name="Assign-Message-1">
```

The following attributes are common to all policy parent elements.

Attribute	Description	Default	Presence
name	The internal name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the <DisplayName> element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Required
continueOnError	Set to false to return an error when a policy fails. This is expected behavior for most policies. Set to true to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to true to enforce the policy. Set to false to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to true, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to true improves API proxy performance. However, overusing async can hurt performance with too much thread switching.	false	Optional

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.4.3.2 <DisplayName> element

Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

<DisplayName>Policy Display Name</DisplayName>

Default:	N/A If you omit this element, the the value of the policy'sname attribute is used.
Presence:	Optional
Type:	String

8.7.4.3.3 <AssignTo> element

The <AssignTo> element determines the variable that the Assign Message policy operates on. The options are:

The request object received by the API proxy

The response object returned from the target server

A custom request or response object

For example, if you use the Assign Message policy to add a query parameter, the query parameter is added to the variable specified by the <AssignTo> element.

If the <AssignTo> element is missing, or if the element does not explicitly specify the name of the destination variable, then the policy assigns values to either the request or response object, depending on the Flow to which the policy is attached. For example, if you attach the policy to a request Flow, the default object is the request object.

Attributes

No destination variable:

```
<AssignTo createNew="false" transport="http" type="request"/>
```

Explicit destination variable:

```
<AssignTo createNew="true" transport="http" type="request">destRequestObj</AssignTo>
```

Attribute	Description	Default	Presence	Type
createNew	If createNew is set to true, then a new variable of the type specified by type is created. If you do not specify the name of the new variable, then the policy creates a	If createNew is not specified, the poplicy responds in one of two ways:	Optional	Boolean

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

	new request or response object, based on the value of type. Caution: When creating a new request orresponse object, the existing request orresponse object is deleted and replaced by the new one, meaning all information in the object is lost. If createNew is false, then the policy responds in one of two ways: If AssignTo can resolve the variable name to a message, then it continues processing. For example, if the policy is in a request flow, the variable is the request object. If the policy is in a response, the variable is the responseobject. If AssignTo cannot be resolved, or resolves to a non-message type, then the policy throws an error.	If AssignTo resolves to a message, then processing advances to the next step. If AssignTo cannot be resolved, or resolves to a non-message type, a new variable of type specified in type is created.		
transport	Specifies the transport type for the request or response message type.	http (the only supported value)	Optional	String
type	Specifies the type of the variable. Valid values: request or response	request	Optional	String

8.7.4.3.4 <Copy> element

Copies information from the message specified by the source attribute to the message object specified by this policy's <AssignTo> element.

```
<Copy source="request">
    <Headers/>
    <QueryParams/>
    <FormParams/>
    <Payload> </Payload>
    <Verb/>
    <Version/>
    <StatusCode/>
    <ReasonPhrase/>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        <Path/>
    </Copy>
```

Default: N/A

Presence: Optional

Type: String

Attributes

<Copy source="request">

Attribute	Description	Presence	Type
source	Specifies the source object of the copy. If source is not specified, it is treated as a simple message. For example, if the policy is in the request flow, then the source defaults to the request object. If the policy is in the response flow, it defaults to the response object. If you omit source, you can use an absolute reference to a flow variable as the source of the copy. For example, specify the value as {request.header.user-agent}. If the source variable cannot be resolved, or resolves to a non-message type, <Copy> fails to respond.	Optional	String

<Copy>/<Headers> element

Copies the specified HTTP header from the source to the message variable specified with the <AssignTo> element. To copy all headers, specify<Copy><Headers/></Copy>.

```
<Copy source='request'>
    <Headers>
        <Header name="headerName"/>
    </Headers>
</Copy>
```

If there are multiple headers with the same name, use the following syntax:

```
<Copy source='request'>
    <Headers>
      <Header name="h1"/>
      <Header name="h2"/>
      <Header name="h3.2"/>
    </Headers>
</Copy>
```

This example copies "h1", "h2", and the second value of "h3". If "h3" has only one value, then it is not copied.

Default: N/A

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Presence: Optional

Type: String

<Copy> / <QueryParams> element

Copies query parameters. Note that the QueryParams is copied only when both the source and AssignTo type are request. To copy all query parameters, specify <Copy><QueryParams/></Copy>. This example copies the user-agent header to the message variable specified with the<AssignTo> element:

```
<Copy source='request'>
    <QueryParams>
        <QueryParam name="paramName"/>
    </QueryParams>
</Copy>
```

If there are multiple query params with the same name, use the following syntax:

```
<Copy source='request'>
    <QueryParams>
      <QueryParam name="qp1"/>
      <QueryParam name="qp2"/>
      <QueryParam name="qp3.2"/>
    </QueryParams>
</Copy>
```

This example copies "qp1", "qp2", and the second value of "qp3". If "qp3" has only one value, then it is not copied.

Default: N/A

Presence: Optional

Type: String

<Copy> / <FormParams> element

Copies the form parameters. Note that the FormParams is copied only when the contentType is source and AssignTo is application/x-www-form-urlencoded. To copy all form parameters, specify <Copy><FormParams/></Copy>. This example copies specified form parameter from the object specified in the source attribute to the variable specified in the <AssignTo> element:

```
<Copy source='request'>
    <FormParams>
        <FormParam name="paramName"/>
    </FormParams>
</Copy>
```

If there are multiple form params with the same name, use the following syntax:

```
<Copy source='request'>
    <FormParams>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
      <FormParam name="f1"/>
      <FormParam name="f2"/>
      <FormParam name="f3.2"/>
    </FormParams>
</Copy>
```

This example copies "f1", "f2", and the second value of "f3". If "f3" has only one value, then it is not copied.

Default:	N/A
Presence:	Optional
Type:	String

<Copy> / <Payload> element

If true, the Content-Type header is copied from the object specified in the source attribute to the variable specified in the <AssignTo> element.

```
<Copy source='request'>
    <Payload>true</Payload>
</Copy>
```

Default:	false
Presence:	Optional
Type:	Boolean

<Copy> / <Version> element

If true, the HTTP version is copied from the object specified in the source attribute to the variable specified in the <AssignTo> element.

```
<Copy source='request'>
    <Version>true</Version>
</Copy>
```

Default:	false
Presence:	Optional
Type:	Boolean

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

<Copy> / <Verb> element

If true, copies the verb found in the source attribute object to the variable specified in the <AssignTo> variable. The verb is assigned only when both the source and AssignTo type are request.

```
<Copy source='request'>
    <Verb>true</Verb>
</Copy>
```

Default:	false
Presence:	Optional
Type:	Boolean

<Copy> / <Path> element

Copies the path found in the source attribute object to the variable specified in the <AssignTo> variable. The path is assigned only when both the source and AssignTo type are request.

```
<Copy source='request'>
    <Path>true</Path>
</Copy>
```

Default:	false
Presence:	Optional
Type:	Boolean

<Copy> / <StatusCode> element

If true, copies the StatusCode from the object specified by the source attribute to the variable specified in the AssignType element. The StatusCodeis copied only when both the source and AssignTo type are response.

```
<Copy source='request'>
    <StatusCode>true</StatusCode>
</Copy>
```

Default:	false
Presence:	Optional
Type:	Boolean

<Copy> / <ReasonPhrase> element

If true, copies the ReasonPhrase from the object specified by the source attribute to the variable specified in the AssignType element. The ReasonPhrase is copied only when both the source and AssignTo type are response.

```
<Copy source='request'>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <ReasonPhrase>true</ReasonPhrase>
</Copy>
```

Default:	false
Presence:	Optional
Type:	Boolean

8.7.4.3.5 <Remove> element

Removes specified elements from the message variable specified in the <AssignTo> element of this policy. For example, a common use case is to remove a query parameter that contains sensitive information from the incoming request object, to avoid passing it to the backend server.

```
<Remove>
      <Headers>
          <Header name="h1"/>
      </Headers>
      <QueryParams>
          <QueryParam name="q1"/>
      </QueryParams>
      <FormParams>
          <FormParam name="f1"/>
      </FormParams>
      </Payload>
  </Remove>
```

Default:	N/A
Presence:	Optional
Type:	N/A

<Remove>/<Headers> element

Removes specified HTTP headers from the message variable specified in the <AssignTo> element of this policy. To remove all the headers, specify<Remove><Headers/></Remove>. This example removes the user-agent header from the message variable specified with the <AssignTo>element.

```
<Remove>
    <Headers>
        <Header name="user-agent"/>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    </Headers>
</Remove>
```

If there are multiple headers with the same name, use the following syntax:

```
<Remove>
    <Headers>
      <Header name="h1"/>
      <Header name="h2"/>
      <Header name="h3.2"/>
    </Headers>
</Remove>
```

This example removes "h1", "h2", and the second value of "h3". If "h3" has only one value, then it is not removed.

Default:	N/A
Presence:	Optional
Type:	String

<Remove>/<QueryParams> element

Removes specified query parameters from the message variable specified in the <AssignTo> element of this policy. To remove all query paremeters, specify <Remove><QueryParams/></Remove>.

Note: This option only works when the <AssignTo> type is request.

```
<Remove>
    <QueryParams>
        <QueryParam name="param_name"/>
    </QueryParams>
</Remove>
```

If there are multiple query params with the same name, use the following syntax:

```
<Remove>
    <QueryParams>
      <QueryParam name="qp1"/>
      <QueryParam name="qp2"/>
      <QueryParam name="qp3.2"/>
    </QueryParams>
</Remove>
```

This example removes "qp1", "qp2", and the second value of "qp3". If "qp3" has only one value, then it is not removed.

Default:	N/A

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Presence:	Optional
Type:	String

<Remove>/<FormParams> element

Removes specified query parameters from the message variable specified in the <AssignTo> element of this policy. To remove all form parameters, specify <Remove><FormParams/></Remove>.

Note: This option only works when the contentType of <AssignTo> element is application/x-www-form-urlencoded.

```
<Remove>
    <FormParams>
        <FormParam name="form_param_name"/>
    </FormParams>
</Remove>
```

If there are multiple form params with the same name, use the following syntax:

```
<Remove>
    <FormParams>
      <FormParam name="f1"/>
      <FormParam name="f2"/>
      <FormParam name="f3.2"/>
    </FormParams>
</Remove>
```

This example removes "f1", "f2", and the second value of "f3". If "f3" has only one value, then it is not removed.

Default:	N/A
Presence:	Optional
Type:	String

<Remove>/<Payload> element

If true, the payload is cleared from the message variable specified in the <AssignTo> element.

```
<Remove>
  <Payload>true</Payload>
</Remove>
```

Default:	false
Presence:	Optional
Type:	Boolean

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.4.3.6 <Add> element

Adds information to the message object specified with this policy's <AssignTo> element.

```
<Add>
    <Headers>
        <Header name="h1">value</Header>
    </Headers>
    <QueryParams>
        <QueryParam name="q1">value</QueryParam>
    </QueryParams>
    <FormParams>
        <FormParam name="f1">value</FormParam>
    </FormParams>
</Add>
```

Default:	N/A
Presence:	Optional
Type:	String

<Add>/<Headers> element

Adds the headers in the variable specified in the <AssignTo> element. Note that the empty header <Add><Headers/></Add> does not add any header. The same holds true for QueryParams and FormParams. This example adds the user-agent header to the message variable specified with the <AssignTo> element, and copies the value of the request.user.agent flow variable into the header.

```
<Add>
    <Headers>
        <Header name="user-agent">{request.user.agent}</Header>
    </Headers>
</Add>
```

Default:	N/A
Presence:	Optional
Type:	String

<Add>/<QueryParams> element

Adds specified query parameters to the message variable specified in the <AssignTo> element of this policy. Note that the empty query params, <Add><QueryParams/></Add>, does not add any query params.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Note: This option only works when the <AssignTo> type is request.

This example adds the query param named "param_name" and assign the value 27 to it:

```
<Add>
    <QueryParams>
        <QueryParam name="param_name">27</QueryParam>
    </QueryParams>
</Add>
```

Default:	N/A
Presence:	Optional
Type:	String

<Add>/<FormParams> element

Adds specified form parameters to the message variable specified in the <AssignTo> element of this policy. Note that the empty form params, <Add><FormParams/></Add> does not add any form params.

Note: This option only works when the contentType of <AssignTo> element is application/x-www-form-urlencoded.

This example adds the specified form parameter to the message variable specified with the <AssignTo> element, and assigns it the value of the "name" query param:

```
<Add>
    <FormParams>
        <FormParam name="{my_username}">{request.queryparam.name}</FormParam>
    </FormParams>
</Add>
```

Default:	N/A
Presence:	Optional
Type:	String

8.7.4.3.7 <Set> element

Sets information in the message object specified in with this policy's <AssignTo> element.

```
    <Set>
        <Headers/>
        <QueryParams/>
        <FormParams/>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        <Payload> </Payload>
        <Verb/>
        <StatusCode/>
        <ReasonPhrase/>
        <Path/>
    </Set>
```

Default:	N/A
Presence:	Optional
Type:	N/A

<Set>/<Headers> element

Sets or overwrites HTTP headers in the message variable specified with the <AssignTo> element. Note that the empty header <Set><Headers/></Set> does not set any header. This example sets the user-agent header to the message variable specified with the <AssignTo> element.

```
<Set>
    <Headers>
        <Header name="user-agent">{request.header.user-agent}</Header>
    </Headers>
</Set>
```

Default:	N/A
Presence:	Optional
Type:	String

<Set>/<QueryParams> element

Sets query parameters. Note that the empty query parameter <Set><QueryParams/></Set> does not set any query parameters. This example assigns a flow variable to a query parameter called "address" to the HTTP message specified with the <AssignTo> element.

```
<Set>
    <QueryParams>
        <QueryParam name="address">{request.queryparam.address}</QueryParam>
    </QueryParams>
</Set>
```

Default:	N/A
Presence:	Optional
Type:	String

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

<Set>/<FormParams> element

Sets/overwrites the form parameters and the contentType of message changes to application/x-www-form-urlencoded. Note that the empty form parameter <Set><FormParams/></Set> does not set any form parameters. This example sets the specified form parameter to the object specified in the source attribute to the variable specified in the <AssignTo> element.

```
<Set>
    <FormParams>
        <FormParam name="myparam">{request.formparam.myparam}</FormParam>
    </FormParam>
</Set>
```

Default:	N/A
Presence:	Optional
Type:	String

<Set>/<Payload> element

Set a plain text payload:

```
<Set>
    <Payload contentType="text/plain">test1234</Payload>
</Set>
```

Set a JSON payload. Notice how this example escapes the leading "{" in the payload. Otherwise, system would interpret the leading "{" as the beginning of a variable reference:

```
<Set>
    <Payload contentType="application/json">
        \{"name":"foo", "type":"bar"}
    </Payload>
</Set>
```

Reference a variable in a JSON payload. You cannot use curly braces to reference variables in a JSON payload. Instead, you must declare and embed the variable delimiter characters, as follows:

```
<Set>
    <Payload contentType="application/json" variablePrefix="@" variableSuffix="#">
        {"name":"foo", "type":"@variable_name#"}
    </Payload>
</Set>
```

Set a mixed payload in XML:

```
<Set>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <Payload contentType="text/xml">
        <root>
          <e1>sunday</e1>
          <e2>funday</e2>
          <e3>{var1}</e3>
    </Payload>
</Set>
```

Default:

Presence: Optional

Type: String

Attributes

<Payload contentType="content_type" variablePrefix="char" variableSuffix="char">

Attribute	Description	Presence	Type
contentType	If contentType is specified, its value is assigned to the contentType header.	Optional	String
variablePrefix	Optionally specifies the leading delimiter on a flow variable because JSON payloads cannot use the default "{" character.	Optional	Char
variableSuffix	Optionally specifies the trailing delimiter on a flow variable because JSON payloads cannot use the default "}" character.	Optional	Char

<Set> / <Version> element

Sets the version. Note that the content of <Version> is treated as message template. The same holds true for Verb, Path, StatusCode, and ReasonPhrase.

```
<Set>
    <Version>2.1</Version>
</Set>
```

Default: false

Presence: Optional

Type: Boolean

<Set> / <Verb> element

Sets the HTTP verb only when AssignTo type is request.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<Set>
    <Verb>POST</Verb>
</Set>
<Set>
    <Verb>${variable.v1}</Verb>
</Set>
```

Default:	false
Presence:	Optional
Type:	String (a valid HTTP verb)

<Set>/<Path> element

Sets the HTTP path of the request only when AssignTo type is request. Specify the path relative to the domain, which is defined by the target endpoint. For example, to set the request path as http://myCo.com/a/b/c, where http://myCo.com is defined by the API proxy's target endpoint, specify the path as:

```
<Set>
    <Path>/a/b/c</Path>
</Set>
```

Default:	
Presence:	Optional
Type:	String

<Set>/<StatusCode> element

Sets the status code and reason phrase only when AssignTo type is response.

```
<Set>
    <StatusCode>400 Bad Request</StatusCode>
</Set>
```

Default:	NA
Presence:	Optional
Type:	String

<Set>/<ReasonPhrase> element

Sets the status code and reason phrase only when AssignTo type is response.

```
<Set>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <ReasonPhrase>true</ReasonPhrase>
</Set>
```

Default:	false
Presence:	Optional
Type:	Boolean

8.7.4.3.8 <AssignVariable> element

Lets you assign a value to a flow variable. This example assigns the value of the flow variable request.header.user-agent to the flow variablemyvar and the value of the query parameter country to the Country variable:

```
<AssignVariable>
    <Name>myvar</Name>
    <Ref>request.header.user-agent</Ref>
    <Value>ErrorOnCopy</Value>
</AssignVariable>
<AssignVariable>
    <Name>Country</Name>
    <Ref>request.queryparam.country</Ref>
    <Value>ErrorOnCopy</Value>
</AssignVariable>
```

If the variable referenced by the <Ref> tag is not accessible, use the value specified by the <Value> tag.

One way to use <AssignVariable> is to set the default value of a query parameter, header, or other value that can be passed in with the request. For example, you create a weather API proxy where the request takes a single query parameter named "w". This parameter contains the ID of the city for which you want the weather. The request URL has the form:

```
http://myCO.com/v1/weather/forecastrss?w=12797282
```

To define a default value for the "w" query parameter, create an Assign Message policy as shown below:

```
<AssignMessage async="false" continueOnError="false" enabled="true" name="DefaultWQP">
  <AssignTo createNew="false" transport="http" type="request"/>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
  <AssignVariable>
    <Name>request.queryparam.w</Name>
    <Ref>request.queryparam.w</Ref>
    <Value>12797282</Value>
  </AssignVariable>
</AssignMessage>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

In this example, you use <AssignVariable> to copy the request.queryparam.w flow variable to itself. If the flow variable is null, meaning the "w" query parameter was omitted from the request, then it copies the default value from the <Value> element. Therefore, you can make a request to this API proxy that omits the "w" query parameter:

```
http://myCO.com/v1/weather/forecastrss
```

And still have the API proxy return a valid result.

<AssignVariable> / <Name> element

Specifies the name of the destination flow variable. If the variable does not exist, a new one is created.

<AssignVariable> / <Ref> element

Specifies the source of the assignment as a flow variable or other value. If you specify a flow variable, omit the enclosing brackets "{}" normally used to reference a flow variable.

<AssignVariable> / <Value> element

If <Ref> is not specified or is unresolvable or null, this value is assigned to the <Name> variable. The value is always interpreted as a literal string; you cannot use a flow variable or other variable as the value.

8.7.4.4 Flow variables

This policy does not automatically create or populate any flow variables. However, you can create and populate flow variables manually using this policy's <AssignTo>, <Copy>, <Set>, <Add>, and <AssignVariable> elements.

8.7.4.5 Error codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

The Assign Message policy type defines the following error codes.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Error Code	Message
UnresolvedVariable	AssignMessage[{0}]: unable to resolve variable {1}
VariableOfNonMsgType	AssignMessage[{0}]: value of variable {1} is not of type Message
SetVariableFailed	AssignMessage[{0}]: failed to assign message to {1}
InvalidIndex	AssignMessage[{0}]: index must be greater than zeroin {1}
InvalidVariableName	AssignMessage schema validation failed: invalid variable name - {0} - in assign variable
InvalidPayload	Invalid payload {0}

8.7.5 Persist Data Using KeyValueMap

The KeyValueMapOperations policy provides policy-based access to a key/value store available in SAP API Management. The store provides a lightweight persistence mechanism for data formatted as key/value pairs, which can be accessed at runtime by policies or code running on SAP API Management. KeyValueMapOperations are typically used to store or retrieve long-lived information that needs to be reused over multiple request/response transactions.

KeyValue refers to any arbitrary data in the format `key=value`, for example `localhost=127.0.0.1`,`zip_code=94110`, or `first_name=felix`.

In the first example, localhost is a key, and 127.0.0.1 is a value.

Each key/value pair is stored in a map as an entry. A key/value map can store many entries.

For example, say you need to store a list of IP addresses associated with various backend environments. You could create a map called ipAddresses that contains a list of key/value pairs as entries. For example, this JSON might represent such a map:

```
{
  "entry" : [ {
    "name" : "Development",
    "value" : "65.87.18.18"
  }, {
    "name" : "Staging",
    "value" : "65.87.18.22"
  } ],
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
  "name" : "ipAddresses"
}
```

You could use this structure to create a store of IP addresses that could be used by policies at runtime to enforce IP whitelisting or blacklisting, to dynamically select a backend target address, and so on.

KeyValue pairs can be stored, retrieved and deleted from named maps by configuring policies that specify PUT, GET, or DELETE operations.

The following operations are supported in the KeyValueMapOperations policy.

- PUT: Writes entries to an existing KeyValueMap
- GET: Retrieves the value for the key specified
- DELETE: Deletes the key or value specified

> **Note**
>
> Key/Value maps can be manipulated via the KeyValueMapOperations policy, or directly via the SAP API Management management API.

8.7.5.1 Samples

A simple example of a useful KeyValueMap is a URL 'shortening' service. The KeyValueMap could be configured to store shortened URLs along with corresponding full URLs.

KeyValueMap PUT Sample

The following policy configuration creates a KeyValueMap. The policy PUTs a key with two associated values into a key/value map named "urlMapper".

```
<KeyValueMapOperations name="putUrl" mapIdentifier="urlMapper">
  <Scope>apiproxy</Scope>
  <Put override="true">
    <Key>
      <Parameter ref="urlencoding.requesturl.hashed"/>
    </Key>
    <Value ref="urlencoding.longurl.encoded"/>
    <Value ref="request.queryparam.url"/>
  </Put>
</KeyValueMapOperations>
```

The key in this example, urlencoding.requesturl.hashed, is an example of a custom variable. The hashed request URL would would be generated by code (JavaScript for example) and then stored in this variable, where the KeyValueMapOperations policy can access it.

For each key, requesturl.hashed, two values are stored:

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- The contents of the custom variable named urlencoding.longurl.encoded
- The contents of the pre-defined variable request.queryparam.url

When the policy executes at runtime, the values of the variables are as follows:

- urlencoding.requesturl.hashed: ed24e12820f2f900ae383b7cc4f2b31c402db1be
- urlencoding.longurl.encoded: http://tinyurl.com/38lwmlr
- request.queryparam.url: http://sap.com

The following key/value map and entry would be generated in SAP API Management 's key/value store and scoped to the API proxy to which the policy is attached:

```
{
  "entry" : [ {
    "name" : "ed24e12820f2f900ae383b7cc4f2b31c402db1be",
    "value" : "http://tinyurl.com/38lwmlr,http://sap.com"
  } ],
  "name" : "urlMapper"
}
```

The entry will persist until it is deleted. Key/value store entries are distributed across instances of SAP API Management that are running the cloud.

KeyValueMap GET Sample

To retrieve the value of key/value map entry, configure a policy to GET the KeyValueMap, as follows:

```
<KeyValueMapOperations name="getUrl" mapIdentifier="urlMapper">
<Scope>apiproxy</Scope>
  <Get assignTo="urlencoding.shorturl" index='1'>
    <Key>
      <Parameter ref="urlencoding.requesturl.hashed"/>
    </Key>
  </Get>
</KeyValueMapOperations>
```

When this policy is executed, if the value of the urlencoding.requesturl.hashed variable is ed24e12820f2f900ae383b7cc4f2b31c402db1be, then the custom variable named urlencoding.shorturl will be set with the value.

Now that the data has been retrieved, other policies and code can access it by extracting the value of those variables.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.5.2 Configuring the KeyValueMapOperation Policy

Configure the KeyValueMap Operation policy using the following elements.

Field Name	Description	Default	Required
mapIdentifier	Specifies an identifier to be used when accessing, via APIs, a map created by this policy. By default, this policy can be accessed through APIs via the name "kvmap". Within a scope of organization/environment/apiproxy, you can use the mapIdentifierattribute to specify your own map name.	N/A	Optional.

Field Name	Description	Default	Required
name	The unique name of the policy. Use this name to reference the policy within a Step definition element. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the `<DisplayName>` element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Yes
continueOnError	Set to `false` to return an error when a policy fails. This is expected behavior for most policies. Set to `true` to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to `true` to enforce the policy. Set to `false` to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to `true`, policy execution is offloaded to a different thread, leaving the	false	Optional

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name	Description	Default	Required
	main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to `true` improves API proxy performance. However, overusing async can hurt performance with too much thread switching. To use asynchronous behavior in API proxies, see Java Callout Policy.		

8.7.5.2.1 <DisplayName> element

Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

```
<DisplayName>Policy Display Name</DisplayName>
```

Default: N/A
If you omit this element, the the value of the policy'sname attribute is used.

Presence: Optional

Type: String

8.7.5.2.2 <Delete> element

Deletes the specified key/value pair. (At least one of `<Get>, <Put>, or <Delete>` must be used.)

```
<Delete>
   <Key>
      <Parameter>key_name_literal</Parameter>
   </Key>
</Delete>
```

Default: N/A

Presence: Required if <Get> or <Put> are not present.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Type: N/A

8.7.5.2.3 <DisplayName> element

A natural-language name that labels the policy in the management UI proxy editor. If omitted, the policy name attribute is used.

```
<DisplayName>Key Value Map Operations 1</DisplayName>
```

Default: Policy name attribute value.

Presence: Optional

Type: String

8.7.5.2.4 <Entry> element

Seed values for KeyValueMaps, which are populated in the KeyValueMap when it is initialized.

```
<InitialEntries>
   <Entry>
      <Key>
         <Parameter>key_name_literal</Parameter>
      </Key>
      <Value>v1</Value>
   </Entry>
   <Entry>
      <Key>
         <Parameter>key_name_variable</Parameter>
      </Key>
      <Value>v3</Value>
      <Value>v4</Value>
   </Entry>
</InitialEntries>
```

Default: N/A

Presence: Optional

Type: N/A

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.5.2.5 <ExclusiveCache> element

Deprecated. Use the <Scope> element instead.

8.7.5.2.6 <ExpiryTimeInSecs> element

For a GET from the KeyValueMap, specifies the duration in seconds after which SAP API Management refreshes its cached value from the KeyValueMap.

They Key Value Map Operations policy lets you determine how long values are persisted before being refreshed. The refresh interval is set with the <ExpiryTimeInSecs> element. If a GET operation is executed and the expiry interval has been exceeded, the value is refreshed and the policy gets the updated value. When you add this policy to an API proxy, the default expiry time is now 300 seconds.

8.7.5.2.7 <Get> element

Retrieves the value for the key specified. (At least one of <Get>, <Put>, or <Delete> must be used.)

```
<Get assignTo="myvar" index="1">
   <Key>
      <Parameter>key_name_literal</Parameter>
   </Key>
</Get>
```

Default: N/A

Presence: Required if <Put> or <Delete> are not present.

Type: N/A

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.5.2.8 Attributes

Attribute	Description	Default	Presence
assignTo	The variable to which the retrieved value should be assigned.	N/A	Required
index	The index number (in a 1-based index) of the item to fetch from a multi-valued key. For example, specifying index=1 will return the first value and assign it to the assignTovariable. If no index value is specified, all the values of that entry are assigned to the variable as a java.util.List. For an example, see the KeyValueMapOperations GET tab.	N/A	Optional

8.7.5.2.9 <InitialEntries> element

Seed values for KeyValueMaps, which are populated in the KeyValueMap when it is initialized.

```
<InitialEntries>
   <Entry>
      <Key>
         <Parameter>key_name_literal</Parameter>
      </Key>
      <Value>v1</Value>
   </Entry>
   <Entry>
      <Key>
         <Parameter>key_name_variable</Parameter>
      </Key>
      <Value>v3</Value>
      <Value>v4</Value>
   </Entry>
</InitialEntries>
```

Default: N/A

Presence: Optional

Type: N/A

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.5.2.10 <Key> element

Specifies the key in a key/value map entry. A key can be composite, which means that more than one parameter can be appended to create the key. For example, userID and role might be combined to create a key.

The name ratelimit is reserved.

```
<Key>
    <Parameter>key_name_literal</Parameter>
</Key>
```

Default: N/A

Presence: Optional

Type: N/A

8.7.5.2.11 <Parameter> element

Specifies the key name in a key/value pair. This element specifies the name when creating, putting, retrieving, or deleting the key/value pair.

A key can be composite, which means that more than one parameter can be appended to make the key. For example, userIDand role might be combined constitute a key.

You can specify the name as either a literal string or, using the ref attribute, as a variable to be retrieved at run time.

```
<!-- Specify a literal value -->
<Parameter>literal<Parameter>
```

OR

```
<!-- Specify the name of variable value to be populated at run time. -->
<Parameter ref="variable_name"><Parameter>
```

Whether you're getting, updating, or deleting a key/value entry, remember that you need to specify the key name in the same way you did when you created the entry. For example, if you created the key name from a variable, you must use the same variable to get it.

Default: N/A

Presence: Required

Type: String

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.5.2.12 Attributes

Attribute	Description	Default	Presence
ref	Specifies the name of a variable whose value should be used for the key name.	N/A	Required if no literal value is given between the opening and closing tags. Prohibited if a literal value is given.

8.7.5.2.13 <Put> element

Writes a key/value pair to an existing key value map. (At least one of <Get>, <Put>, or <Delete> must be used.)

```
<Put override="false">
   <Key>
      <Parameter ref="mykeyvar"></Parameter>
   </Key>
   <Value ref="myvalvar1"/>
</Put>
```

Default: N/A

Presence: Required if <Get> or <Delete> are not present.

Type: N/A

8.7.5.2.14 Attributes

Attribute	Description	Default	Presence
override	If set to true, it overrides the value for a key.	false	Optional

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.5.2.15 <Scope> element

Defines the boundary of accessibility for KeyValueMaps. The default scope is environment, meaning that, by default, maps entries are shared by all API proxies running in an environment (for example, test or prod). If you set the scope to apiproxy, then entries in the KeyValueMap are accessible only by the API proxy that writes the values to the map.

Note that when accessing a map or map entry, you must specify the same scope value you used when the map was created. For example, if the map was created with a scope of apiproxy, you must use the apiproxy scope when retrieving its values, putting changes, or deleting entries.

```
<Scope>environment</Scope>
```

Default:	environment
Presence:	Optional
Type:	String
Valid values:	organization environment apiproxy policy

8.7.5.2.16 <Value> element

Specifies the value in a key/value pair.

You can specify the value as either a literal string or, using the ref attribute, as a variable to be retrieved at run time.

```
<!-- Specify a literal value -->
<Value>literal<Value>
```

OR

```
<!-- Specify the name of variable value to be populated at run time. -->
<Value ref="variable_name"><Value>
```

You can also include multiple <Value> elements to specify a multi-part value. Values are combined at run time.

In the example below, the map is initialized by the two key/value entries [k1] -> [v1, v2] and [k2, k3] -> [v3,v4].

```
<InitialEntries>
   <Entry>
      <Key>
         <Parameter>k1</Parameter>
      </Key>
      <Value>v1</Value>
      <Value>v2</Value>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
   </Entry>
   <Entry>
      <Key>
         <Parameter>k2</Parameter>
      </Key>
      <Key>
         <Parameter>k3</Parameter>
      </Key>
      <Value>v3</Value>
      <Value>v4</Value>
   </Entry>
</InitialEntries>
```

Default: N/A

Presence: Required

Type: String

8.7.5.3 Policy-specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

The KeyValueMap Policy type defines the following error codes:

Error Code	Message
SetVariableFailed	Failed to set variable {0} in KeyValueMapStepDefinition {1}
RemoveVariableFailed	Failed to remove variable {0} in KeyValueMapStepDefinition{1}
InvalidIndex	Invalid index {0} in KeyValueMapStepDefinition {1}
KeyIsMissing	Key element is not missing in KeyValueMapStepDefinition {0}
ValueIsMissing	Value element is missing in KeyValueMapStepDefinition {0}

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.6 Retrieve Entity Profiles Using AccessEntity

SAP API Management stores profile data for a range for "entities", such as developers, apps, and API products. The AccessEntity policy enables developers to retrieve those profiles during API proxy message processing. As such, the AccessEntity policy functions as a policy-based runtime database lookup. The profile information returned by this policy can be used to enable dynamic behavior, such as conditional endpoint routing, flow execution, policy enforcement, and so on.

For example, you could use the AccessEntity policy to get the profile for an app, and then extract a custom field (such as a department name) from the app profile. Using the department name as a variable, you could route the request to a specific backend service, or you could forward the department name to Analytics to enable data accounting.

When a policy of type AccessEntity is enforced:

1. The policy sets an entity as an XML-formatted flow variable. The variable that is set is usually consumed by an ExtractVariable or AssignMessage policy, as illustrated in Extract message content using ExtractVariables and Generate or modify messages using AssignMessage.
2. XPath is used to parse the desired properties from the profile. For examples see Retrieving entity profiles using the Platform API.
3. If the specified entity is not found, the policy returns ResourceNotFoundException.

AccessEntity can be used to access profiles for the following entities:

- App
- API product
- Company
- Company developer
- Consumer key
- Developer

The basic of structure of an AccessEntity policy is:

```
<AccessEntity name="{policy_name}">
  <EntityType  value="{entity_type}"/>
  <EntityIdentifier ref="{entity_identifier}" type="identifer_type"/>
  <SecondaryIdentifier ref="{secondary_identifier}" type="identifer_type"/>
</AccessEntity>
```

You can access multiple entities of the same type as follows:

```
<AccessEntity name="name_of_the_policy">
  <EntityType  value="type_of_entity"/>
  <Identifiers>
    <Identifier>
      <EntityIdentifier ref="reference_to_entity_identifier" type*=”identifer_type”/>
      <SecondaryIdentifier ref="reference_to_secondary_entity_identifier"
type="identifer_type"/><!-- optional -->
    </Identifier >
    <Identifier>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
      <EntityIdentifier ref="reference_to_entity_identifier" type*=”identifer_type”/>
      <SecondaryIdentifier ref="reference_to_secondary_entity_identifier"
type="identifer_type"/><!-- optional -->
    </Identifier >
  </Identifiers>
</AccessEntity>
```

Where multiple matches exist for a given EntityIdentifier and SupportingIdentifier combination, the first match will be populated in the Flow. An example of such a potential conflict is retrieving anapiproduct based on appid. For example:

- EntityType is set to apiproduct
- EntityIdentifier is set to appid

Since an app can be associated with multiple API products, only the first matched API product will be available in the Flow.

To resolve such potential conflicts, you can use a SecondaryIdentifier. For example, during an OAuth 2.0 exchange, appname and developerid variables are populated by default in the Flow. Those two variables can therefore be used by an AccessEntity policy to get profile details on the requesting app. To prevent conflicts, set the EntityIdentifier to appname and developerid as the SecondaryIdentifier.

8.7.6.1 AccessEntity Variables

When the entity profile specified in the AccessEntity policy is retrieved, the XML-formatted profile object is added to the message context as a variable. It can be accessed like any other variable, with reference to the variable name. The user-provided name of the AccessEntity policy is set as the variable prefix of the variable name.

For example, if an AccessEntity policy with name of GetDeveloper is executed, then the XML-formatted profile is stored in the variable named AccessEntity.GetDeveloper. The XML-formatted profile can then be parsed using an XPath defined in an ExtractVariables policy that specifies AccessEntity.GetDeveloper as its source.

8.7.6.2 EntityType Specification

EntityType	EntityIdentifiers	SecondaryIdentifiers
apiproduct	· apiproductname · appname · appid · consumerkey	· developeremail · developerid · companyname · apiresource The SecondaryIdentifiers developeremail, developerid and companyname are used only when the EntityIdentifier is appname. TheSecondaryIdentifier apiresource is used only when the EntityIdentifier is appname, appid, orconsumerkey.
app	· appname	· developeremail

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

EntityType	EntityIdentifiers	SecondaryIdentifiers
	· appid · consumerkey	· developerid · companyname Used only when identifier is appname, because the value of appname is not guaranteed to be unique across an organization.
company	· companyname · appid · consumerkey	None
companydeveloper	companyname	None
consumerkey	consumerkey	None
developer	· developeremail · developerid · appid · consumerkey	None

8.7.6.3 Configuring an AccessEntity Policy

AccessEntity policies expose the following configuration settings:

Name	Description	Default	Required?
AccessEntity	A policy used to retrieve entity profiles from SAP API Management data store.		
Name	The unique name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.	N/A	Yes
EntityType	The name of the type of entity to be retrieved. For a reference of entity types, see the EntityType specification.	N/A	Yes
EntityIdentifier	The value that identifies the specific entity whose profile should be retrieved. The ref attribute identifies the variable that provides the source of the	N/A	No

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Default	Required?
	identifier, for example,request.queryparam.apikey. The type identifies the EntityType populated by the referenced variable, such as consumerkey.		
SecondaryIdentifier	Use when EntityIdentifier is not guaranteed unique, as with appname. Also use to filter queries that return multiple results. The ref attribute identifies the variable that provides the source of the identifier, for example,request.queryparam.apikey. The type identifies the entity type populated by the referenced variable, such as consumerkey. The use of multiple SecondaryIdentifier elements is not supported. This element is optional to support backward-compatibility with existing deployments.	N/A	No

8.7.6.4 Configuring ExtractVariables to Use AccessEntity Lookup

When an AccessEntity policy executes, the result is a Flow variable populated with an XML-formatted profile object. The XML-formatted profile object can be used as a source of data for a Flow by configuring an ExtractVariables policy to parse the profile.

In the following example, an ExtractVariables policy is used to parse a developer profile retrieved by AccessEntity to obtain the value of a custom attribute. The source is set to point to the variable where the developer profile is stored by the AccessEntity policy: AccessEntity.{policy_name}. The profile can be parsed using XPath, just as any other XML document. Sample profiles for entities managed by SAP API Management are provided below.

8.7.6.5 Convert profile to a message

Before the profile can be parsed by SAP API Management, it must be converted into a request message using an AssignMessage policy.

```
<AssignMessage name="ConvertXmlToMessageRequest">
    <DisplayName>ConvertXmlToMessageRequest</DisplayName>
    <Set>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
      <Payload type="text/xml">{AccessEntity.GetDeveloper}</Payload>
    </Set>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
  <AssignTo createNew="true" transport="http"
type="request">accessentity.typed</AssignTo>
</AssignMessage>
```

Once the profile has been converted to an XML message, you can populate the necessary variables using an ExtractVariables, policy. For example:

```
<ExtractVariables name="GetDeveloperAttribute">
  <Source>AccessEntity.GetDeveloper</Source>
  <VariablePrefix>developer_attribute</VariablePrefix>
  <XMLPayload>
    <Variable name="developer_attribute" type="string">
      <XPath>/Developer/Attributes/Name</XPath>
    </Variable>
  </XMLPayload>
</ExtractVariables>
```

8.7.6.6 Retrieving Entity Profiles Using the Management API

When configuring an AccessEntity policy, it can be useful to retrieve sample entity profiles to assist in selection of appropriate identifiers.

For example:

Apps

```
$ curl -H "Accept:text/xml" -X GET \
<host>:<port>/v1/o/{org_name}/apps/{app_id} \
-u myname:mypass
```

Or:

```
$ curl -H "Accept:text/xml" -X GET \
<host>:<port>/v1/o/{org_name}/developers/{developer_email}/apps/{app_name} \
-u myname:mypass
```

Sample profile:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<App name="thomas-app">
    <AccessType>read</AccessType>
    <ApiProducts/>
    <Credentials>
        <Credential>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
            <Attributes/>
            <ConsumerKey>wrqOOOiPArFI0WRoB1gAJMRbOguekJ5w</ConsumerKey>
            <ConsumerSecret>WvOhDrJ8m6kzz7Ni</ConsumerSecret>
            <ApiProducts>
                <ApiProduct>
                    <Name>FreeProduct</Name>
                    <Status>approved</Status>
                </ApiProduct>
            </ApiProducts>
            <Scopes/>
            <Status>approved</Status>
        </Credential>
    </Credentials>
    <AppFamily>default</AppFamily>
    <AppId>ab308c13-bc99-4c50-8434-0e0ed1b86075</AppId>
    <Attributes>
        <Attribute>
            <Name>DisplayName</Name>
            <Value>Tom's Weather App</Value>
        </Attribute>
    </Attributes>
    <CallbackUrl>http://tom.app/login</CallbackUrl>
    <CreatedAt>1362502872727</CreatedAt>
    <CreatedBy>admin@yourcompany.com</CreatedBy>
    <DeveloperId>PFK8IwOeAOW01JKA</DeveloperId>
    <LastModifiedAt>1362502872727</LastModifiedAt>
    <LastModifiedBy>admin@yourcompany.com</LastModifiedBy>
    <Scopes/>
    <Status>approved</Status>
</App>
```

API Product

```
$ curl  -H "Accept:text/xml" -X GET \
<host>:<port>/v1/o/{org_name}/apiproducts/{apiproduct_name} \
-u myname:mypass
```

Sample XPath, retrieves the second API resource (URI) from the API product named weather_free:

```
/ApiProduct['@name=weather_free']/ApiResources/ApiResource[1]/text()
```

Sample profile:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
<ApiProduct name="weather_free">
    <ApiResources>
        <ApiResource>/forecastrss, /reports</ApiResource>
    </ApiResources>
    <ApprovalType>auto</ApprovalType>
    <Attributes>
        <Attribute>
            <Name>description</Name>
            <Value>Introductory API Product</Value>
        </Attribute>
        <Attribute>
            <Name>developer.quota.interval</Name>
            <Value>1</Value>
        </Attribute>
        <Attribute>
            <Name>developer.quota.limit</Name>
            <Value>1</Value>
        </Attribute>
        <Attribute>
            <Name>developer.quota.timeunit</Name>
            <Value>minute</Value>
        </Attribute>
        <Attribute>
            <Name>servicePlan</Name>
            <Value>Introductory</Value>
        </Attribute>
    </Attributes>
    <CreatedAt>1355847839224</CreatedAt>
    <CreatedBy>andrew@yourcompany.com</CreatedBy>
    <Description>Free API Product</Description>
    <DisplayName>Free API Product</DisplayName>
    <Environments/>
    <LastModifiedAt>1355847839224</LastModifiedAt>
    <LastModifiedBy>andrew@yourcompany.com</LastModifiedBy>
    <Proxies/>
    <Scopes/>
</ApiProduct>
```

Company

```
$ curl   -H "Accept:text/xml" -X GET \
<host>:<port>/v1/o/{org_name}/companies/{company_name} \
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
-u myname:mypass
```

Sample profile:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Company name="theramin">
    <Apps/>
    <DisplayName>Theramin Corporation</DisplayName>
    <Organization>theramin-pm</Organization>
    <Status>active</Status>
    <Attributes>
        <Attribute>
            <Name>billing_code</Name>
            <Value>13648765</Value>
        </Attribute>
    </Attributes>
    <CreatedAt>1349208631291</CreatedAt>
    <CreatedBy>andrew@yourcompany.com</CreatedBy>
    <LastModifiedAt>1349208631291</LastModifiedAt>
    <LastModifiedBy>andrew@yourcompany.com</LastModifiedBy>
</Company>
```

Company Developer

```
$ curl -H "Accept:text/xml" -X GET \
<host>:<port>/v1/o/{org_name}/companies/{company_name}/developers/{developer_name} \
-u myname:mypass
```

Sample profile:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Developers>
    <Developer>
        <Email>ntesla@theramin.com</Email>
        <Role>developer</Role>
    </Developer>
</Developers>
```

Consumer key

```
$ curl -H "Accept:text/xml" -X GET \
<host>:<port>/v1/o/{org_name}/developers/{developer_email}/apps/{app_name}/keys/{consumer_key} \
-u myname:mypass
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Sample XPath:

```
/Credential/ApiProducts/ApiProduct[Name='weather_free']/Status/text()
```

Sample profile:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Credential>
    <Attributes/>
    <ConsumerKey>XLotL3PRxNkUGXhGAFDPOr6fqtvAhuZe</ConsumerKey>
    <ConsumerSecret>iNUyEaOOh96KR3YL</ConsumerSecret>
    <ApiProducts>
        <ApiProduct>
            <Name>weather_free</Name>
            <Status>approved</Status>
        </ApiProduct>
    </ApiProducts>
    <Scopes/>
    <Status>approved</Status>
</Credential>
```

Developer

```
$ curl -H "Accept:text/xml" -X GET \
http://<host:port>/v1/o/{org_name}/developers/{developer_email} \
-u myname:mypass
```

Sample XPath:

```
/Developer/Attributes/Attribute[Name='my_custom_attribute']/Value/text()
/Developer/Email/text()
```

Sample profile:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Developer>
    <Apps>
        <App>weatherappx</App>
        <App>weatherapp</App>
    </Apps>
    <Email>ntesla@theramin.com</Email>
    <DeveloperId>4Y4xd0KRZ1wmHJqu</DeveloperId>
    <FirstName>Nikola</FirstName>
    <LastName>Tesla</LastName>
    <UserName>theramin</UserName>
    <OrganizationName>theramin-pm</OrganizationName>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <Status>active</Status>
    <Attributes>
        <Attribute>
            <Name>project_type</Name>
            <Value>public</Value>
        </Attribute>
    </Attributes>
    <CreatedAt>1349797040634</CreatedAt>
    <CreatedBy>rsaha@youcompany.com</CreatedBy>
    <LastModifiedAt>1349797040634</LastModifiedAt>
    <LastModifiedBy>rsaha@yourcompany.com</LastModifiedBy>
</Developer>
```

8.7.6.7 Policy-specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

The AccessEntity Policy type defines the following error codes:

Error Code	Description
InvalidEntityType	Invalid type {0} in ACCESSENTITYStepDefinition {1}

8.7.7 Validate XML and SOAP Using MessageValidation

The MessageValidation Policy type enables you to configure API proxies to reject messages that do not conform to schema (.xsd) or WSDL definition(s). That means developers building apps to consume your API are immediately notified if they are submitting non-conformant requests. Message validation helps developers by indicating where XML tags may not be properly closed, for example. Validation also protects backend services by blocking XML or SOAP messages who structure might cause unpredictable behavior.

Validation can save you a call to tech support, and can help developers avoid time in the forums trying to find troubleshooting help. If developers are notified that they are submitting invalid requests, they can refer to the XML schema for the API to understand how to fix the error. This makes well-understood XML schemas a key component of your API documentation.

SAP API Management enables you to configure your API with a Message Validation policy that validates messages with the XML payload.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

The policy validates that an XML or JSON message is well-formed, and that the SOAP format is correct according to the specified XSD or WSDL file, ensuring that tags are balanced. If the XML configuration is incomplete, the policy returns a message validation error.

8.7.7.1 Samples

```
<MessageValidation name="myPolicy">
   <Source>mymessage</Source>
   <ResourceURL>xsd://sample</ResourceURL>
   <SOAPMessage version="1.1/1.2"/>
   <Element namespace="http://finance.com/1999">PurchaseOrder</Element>
   <Element namespace="http://finance.com/2000">PurchaseOrder</Element>
</MessageValidation>
```

8.7.7.2 Element Reference

The element reference describes the elements and attributes of the MessageValidation policy.

```
<MessageValidation async="false" continueOnError="false" enabled="true" name="SOAP-
Message-Validation-1">
   <DisplayName>SOAP Message Validation 1</DisplayName>
   <Element namespace="http://sample.com">SampleObject</Element>
   <SOAPMessage/>
   <Source>request</Source>
   <ResourceURL>wsdl://SOAP-Message-Validation-1</ResourceURL>
</MessageValidation>
```

8.7.7.3 <MessageValidation> attributes

<MessageValidation async="false" continueOnError="false" enabled="true" name="SOAP-Message-Validation-1">

The following attributes are common to all policy parent elements.

Attribute	Description	Default	Presence
name	The internal name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API	N/A	Required

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Attribute	Description	Default	Presence
	Management Edge UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the *<DisplayName>* element to label the policy in the management UI proxy editor with a different, natural-language name.		
continueOnError	Set to `false` to return an error when a policy fails. This is expected behavior for most policies. Set to `true` to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to `true` to enforce the policy. Set to `false` to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	This attribute is deprecated.	false	Deprecated

8.7.7.4 DisplayName> element

Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

```
<DisplayName>Policy Display Name</DisplayName>
```

Default: N/A

If you omit this element, the the value of the policy'sname attribute is used.

Presence: Optional

Type: String

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.7.5 <Source> element

Identifies the source message to be validated.

- If you do not provide a <Source> value, a value of message is used.
- If the <Source> variable cannot be resolved, or resolves to a non-message type, then one of the following occurs:
  - If the source variable resolves to a null value in the message flow, `asteps.messagevalidation.SourceMessageNotAvailable` error code is thrown.
  - If the source variable resolves to a non-message value, `asteps.messagevalidation.NonMessageVariable` error code is thrown.

```
<Source>mymessage</Source>
```

Default: request

Presence: Optional

Type: String

8.7.7.6 <ResourceURL> element

Identifies the XSD schema or WSDL definition to be used to validate the source message.

If a <ResourceURL> value is not specified, the message is checked for well-formed JSON if the content-type is application/json.

If the WSDL does not have schemas or if the maximum import depth exceeds 10, message validation will fail.

```
<ResourceURL>xsd://sample</ResourceURL>
```

Default: wsdl://<name>

Presence: Optional

Type: String

8.7.7.7 <SOAPMessage> element

Provides the SOAP version against which to validate SOAP messages.

```
<SOAPMessage version="1.1/1.2"/>
```

Default: N/A

Presence: Optional

Type: N/A

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.7.7.1 Attributes

Attribute	Description	Default	Valid values	Presence
version	Identifies the SOAP version against which to validate SOAP messages.	None	1.1 1.2 1.1/1.2	Optional

8.7.7.8 <Element> element

Specifies the root, or parent, element of the messages to be validated.

<Element namespace="http://finance.com/1999">PurchaseOrder</Element>

<Element namespace="http://finance.com/2000">PurchaseOrder</Element>

Default: sampleObject

Presence: Optional

Type: String

8.7.7.8.1 Attributes

Attribute	Description	Default	Presence
namespace	Provides the namespace of the root, or parent, element of the messages to be validated.	`"http://sample.com"`	Optional

8.7.7.9 Configuring the MessageValidation Policy

Configure MessageValidation using the following elements.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Note

The name attribute for this policy is restricted to these characters: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.

Field Name	Description
Source (Optional)	Source message to be validated. · If the source is missing, it is treated as a simple message. For example,<Source>message</Source> · If the source variable cannot be resolved, or resolves to a non-message type, then one of the following occurs: o If the source variable resolves to a null value in the message flow, asteps.messagevalidation.SourceMessageNotAvailable error code is thrown . o If the source variable resolves to a non-message value, asteps.messagevalidation.NonMessageVariable error code is thrown.
ResourceURL (Optional)	XSD or WSDL file used in the schema validation against the source message. The message validation fails if the WSDL file does not have schemas or if the maximum import depth exceeds 10. If ResourceURL is not specified, the message is checked for: · Well-formed XML if the content-type is application/xml · Well-formed JSON if the content-type is application/json
SOAPMessage (Optional)	Verifies that the message corresponds to the SOAP format of the specified version(s). If the version attribute is not specified, the message could correspond to either SOAP 1.1 or 1.2 format (example below).
Element (Optional)	Specifies the valid root elements.

Example

Message Validation policy

```
<MessageValidation name="myPolicy">
    <Source>mymessage</Source>
    <ResourceURL>xsd://sample</ResourceURL>
    <SOAPMessage version="1.1/1.2"/>
    <Element namespace="http://finance.com/1999"> PurchaseOrder</Element>
    <Element namespace="http://finance.com/2000">PurchaseOrder</Element>
</MessageValidation>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.7.10 Policy-specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

The MessageValidation Policy type defines the following error codes:

Error Code	Message
InvalidResourceType	InvalidResourceType MessageValidation {0}: Invalid Resource Type {1}. It should be xsd or wsdl. Context {2}
ResourceCompileFailed	ResourceCompileFailed MessageValidation {0}: Failed to compile resource {1}. Context {2}
RootElementNameUnspecified	RootElementNameUnspecified MessageValidation {0}: RootElement name is not specified
InvalidRootElementName	InvalidRootElementName MessageValidation {0}: RootElement name {1} is invalid
NonMessageVariable	NonMessageVariable Variable {0} does not resolve to a Message
SourceMessageNotAvailable	SourceMessageNotAvailable {0} message is not available for MessageValidation: {1}
NoElements	Resource "{0}" has no element definitions
Failed	MessageValidation {0} failed with reason: "{1}"

8.7.8 JSON Threat Protection policy error codes

This section describes the error messages and flow variables that are set when this policy triggers an error. This information is important to know if you are developing fault rules for a proxy.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.8.1 Runtime Errors

Error name	Fault string	HTTP status	Occurs when
ExceededContainerDepth	JSONThreatProtection[policy_name]:Exceeded container depth at line[line_num]	500	See fault string.
ExceededObjectEntryCount	JSONThreatProtection[policy_name]:Exceeded object entry count at line[line_num]	500	See fault string.
ExceededArrayElementCount	JSONThreatProtection[policy_name]:Exceeded array element count at line[line_num]	500	See fault string.
ExceededObjectEntryNameLength	JSONThreatProtection[policy_name]:Exceeded object entry name length at line [line_num]	500	See fault string.
ExceededStringValueLength	JSONThreatProtection[policy_name]:Exceeded string value length at line[line_num]	500	See fault string.
SourceUnavailable	JSONThreatProtection[policy_name]:Source [var_name] is not available	500	See fault string.
NonMessageVariable	JSONThreatProtection[policy_name]:Variable [var_name] does not resolve to a Message	500	See fault string.
ExecutionFailed	JSONThreatProtection[policy_name]:Execution failed. reason: [string]	500	See fault string.

8.7.8.2 Deployment errors

None.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.8.3 Fault variables

These variables are set when this policy triggers an error.

Variables set	Where	Example
[prefix].[policy_name].failed	[prefix]: jsonattack [policy_name]: The name of the policy to check.	jsonthreatprotection.JTP-SecureRequest.failed= true
fault.[error_name]	[error_name] = The specific error name to check for as listed in the table above.	fault.name Matches "Source Unavailable"

8.7.8.4 Example error response

```
{
  "fault": {
    "faultstring": "JSONThreatProtection[JPT-SecureRequest]: Execution failed. reason:
JSONThreatProtection[JTP-SecureRequest]: Exceeded object entry name length at line 2",
    "detail": {
      "errorcode": "steps.jsonthreatprotection.ExecutionFailed"
    }
  }
}
```

8.7.8.5 Example fault rule

```
<FaultRule name="JSON Threat Protection Policy Faults">
    <Step>
        <Name>AM-CustomErrorResponse</Name>
        <Condition>(fault.name Matches "ExecutionFailed") </Condition>
    </Step>
    <Condition>(jsonattack.JPT-SecureRequest.failed = true) </Condition>
</FaultRule>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.9 XML Threat Protection policy error codes

This section describes the error messages and flow variables that are set when this policy triggers an error. This information is important to know if you are developing fault rules for a proxy.

8.7.9.1 Error code prefix

```
steps.xmlthreatprotection
```

8.7.9.2 Runtime Errors

These errors can occur when the policy executes.

Error name	HTTP status	Cause
NodeDepthExceeded	500	See fault string.
AttrCountExceeded	500	See fault string.
ChildCountExceeded	500	See fault string.
NSCountExceeded	500	See fault string.
ElemNameExceeded	500	See fault string.
AttrNameExceeded	500	See fault string.
AttrValueExceeded	500	See fault string.
NSPrefixExceeded	500	See fault string.
NSURIExceeded	500	See fault string.
PITargetExceeded	500	See fault string.
PIDataExceeded	500	See fault string.
CommentExceeded	500	See fault string.
TextExceeded	500	See fault string.
SourceUnavailable	500	See fault string.
NonMessageVariable	500	See fault string.
ExecutionFailed	500	See fault string.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.9.3 Deployment errors

None.

8.7.9.4 Fault variables

These variables are set when a runtime error occurs

Variables set	Where	Example
[prefix].[policy_name].failed	The [prefix] is xmlattack. The [policy_name] is the name of the policy that threw the error.	xmlattack.XPT-SecureRequest.failed= true
fault.[error_name]	[error_name] = The specific error name to check for as listed in the table above.	fault.name Matches"Source Unavailable"

8.7.9.5 Example error response

```
{
  "fault": {
    "faultstring": "XMLThreatProtection[XPT-SecureRequest]: Execution failed. reason:
XMLThreatProtection[XTP-SecureRequest]: Exceeded object entry name length at line 2",
    "detail": {
      "errorcode": "steps.xmlthreatprotection.ExecutionFailed"
    }
  }
}
```

8.7.9.6 Example fault rule

```
<FaultRule name="XML Threat Protection Policy Faults">
    <Step>
        <Name>AM-CustomErrorResponse</Name>
        <Condition>(fault.name Matches "ExecutionFailed") </Condition>
    </Step>
    <Condition>(xmlattack.XPT-SecureRequest.failed = true) </Condition>
</FaultRule>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.10 Convert XML to JSON

SAP API Management enables developers to convert messages from the extensible markup language (XML) format to JavaScript object notation (JSON) format using the XMLToJSON policy type. This policy is useful for enabling backend XML services to support RESTful apps that require JSON (for example, due to a lack of an XML parsing capabilities on the client).

In a typical mediation scenario, a JSONToXML policy on the inbound request flow is often paired with an XMLToJSON policy on the outbound response flow. By combining policies this way, a JSON API can be exposed for backend services that natively support only XML.

For scenarios where APIs are consumed by diverse clients apps which may require either JSON and XML, the format of the response can be dynamically set by configuring JSONToXML and XMLToJSON policies to execute conditionally. See Flow variables and conditions for an implementation of this scenario.

The HTTP Content-type header of the source message must be set to text/xml. If the HTTP Content-type header of the source message is not text/xml, the policy is not enforced.

The payload of the XML message is parsed and converted into JSON, and the HTTP Content-type header of the XML-formatted message is set to application/json.

8.7.10.1 Sample

This policy type defines a set of reasonable defaults. The minimal policy configuration required to convert XML to JSON is the following:

```
<XMLToJSON name="ConvertToJSON">
  <Options>
  </Options>
  <OutputVariable>response</OutputVariable>
  <Source>response</Source>
</XMLToJSON>
```

Assuming that the intent is to convert an XML-formatted response into a JSON-formatted response, the policy would be attached to a response Flow (for example, the ProxyEndpoint response PostFlow). This configuration takes a JSON-formatted response message as the source, and then creates an XML-formatted message that is populated in the response OutputVariable. SAP API Management automatically uses the content of this variable as the message for next processing step.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.10.2 Configuring an XML to JSON Policy

For more advanced control over mapping XML to JSON structures, use the following configuration elements. All fields are optional. However, if you specify NamespaceBlockName, you must also specifyDefaultNamespaceNodeName and NamespaceSeparator.

8.7.10.2.1 <XMLtoJSON> attributes

```
<XMLtoJSON async="false" continueOnError="false" enabled="true" name="XML-to-JSON-1">
```

Field Name	Description	Default	Required
name	The unique name of the policy. Use this name to reference the policy within a Step definition element. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the `<DisplayName>` element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Yes
continueOnError	Set to `false` to return an error when a policy fails. This is expected behavior for most policies. Set to `true` to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to `true` to enforce the policy. Set to `false` to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	Note: This attribute does not make the policy execute asynchronously. When set to `true`, policy execution is offloaded to a different thread, leaving the main thread free to handle additional requests. When the offline processing is complete, the main thread comes back and finishes handling the message flow. In some cases, setting async to `true` improves API	false	Optional

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name	Description	Default	Required
	proxy performance. However, overusing async can hurt performance with too much thread switching. To use asynchronous behavior in API proxies, see Java Callout Policy.		

8.7.10.2.2 <DisplayName> element

Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

```
<DisplayName>Policy Display Name</DisplayName>
```

Default:	N/A If you omit this element, the the value of the policy'sname attribute is used.
Presence:	Optional
Type:	String

8.7.10.2.3 <Source> element

The variable, request or response, that contains the XML message that you want to convert to JSON.

The HTTP Content-type header of the source message must be set to application/xml, otherwise the policy is not enforced. If <Source> is not defined, then it is treated as message (which resolves to request when the policy is attached to a request flow, or response when the policy is attached to a response flow).

If the source variable cannot be resolved, or resolves to a non-message type, the policy throws an error.

```
<Source>response</Source>
```

Default	request or response, determined by where the policy is added to the API proxy flow
Presence	Optional
Type	message

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.10.2.4 <OutputVariable> element

Stores the output of the XML to JSON format conversion. This is usually the same value as the source, that is, usually XML response is converted to a JSON response.

The payload of the XML message is parsed and converted into JSON, and the HTTP Content-type header of the XML-formatted message is set to application/json.

If OutputVariable is not specified, the source is treated as OutputVariable. For example, if the source isresponse, then OutputVariable defaults to response.

```
<OutputVariable>response</Response>
```

Default	request or response, determined by where the policy is added to the API proxy flow
Presence	Optional
Type	message

8.7.10.2.5 <Options> / <RecognizeNumber> element

If `true`, then number fields in the XML payload retain their original format.

```
<RecognizeNumber>true</RecognizeNumber>
```

Consider the following XML example:

```
<a>
  <b>100</b>
  <c>value</c>
</a>
```

If true, converts to:

```
{
    "a": {
        "b": 100,
        "c": "value"
    }
}
```

If false, converts to:

```
{
    "a": {
        "b": "100",
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        "c": "value"
    }
}
```

Default	false
Presence	Optional
Type	Boolean

8.7.10.2.6 <Options> / <RecognizeBoolean> element

Lets the conversion maintain boolean true / false values rather than turning the values into strings.

```
<RecognizeBoolean>true</RecognizeBoolean>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

For the following XML example:

```
<a>
  <b>true</b>
  <c>value</c>
</a>
```

If `true`, converts to:

```
{
    "a": {
        "b": true,
        "c": "value"
    }
}
```

If `false`, converts to:

```
{
    "a": {
        "b": "true",
        "c": "value"
    }
}
```

Default	false
Presence	Optional
Type	Boolean

8.7.10.2.7 <Options> / <RecognizeNull> element

Lets you convert empty values to null values.

```
<RecognizeNull>true</RecognizeNull>
```

For the following XML:

```
<a>
  <b></b>
  <c>value</c>
</a>
```

If `true`, converts to:

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
{
  "a": {
    "b": null,
    "c": "value"
  }
}
```

If `false`, converts to:

```
{
  "a": {
    "b": {},
    "c": "value"
  }
}
```

Default	false
Presence	Optional
Type	Boolean

8.7.10.2.8 <Options> / <NullValue> element

Indicates what constitutes a null value in the message to be converted. By default the value is `NULL`.

```
<NullValue>NULL</NullValue>
```

Default	NULL
Presence	Optional
Type	String

8.7.10.2.9 <Options> / <NamespaceBlockName> <Options> / <DefaultNamespaceNodeName> <Options> / <NamespaceSeparator> elements

Use these elements together.

```
<NamespaceBlockName>#namespaces</NamespaceBlockName>
<DefaultNamespaceNodeName>&</DefaultNamespaceNodeName>
<NamespaceSeparator>***</NamespaceSeparator>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Consider the following XML example:

```
<a xmlns="http://ns.com" xmlns:ns1="http://ns1.com">
  <ns1:b>value</ns1:b>
</a>
```

If `NamespaceSeparator` is not specified, the following JSON structure is generated:

```
{
    "a": {
        "b": "value"
    }
}
```

If the elements `NamespaceBlockName`, `DefaultNamespaceNodeName`, and `NamespaceSeparator` are specified as#namespaces, &, and ***, respectively, then the following JSON structure is generated:

```
{
    "a": {
        "#namespaces": {
            "&": "http://ns.com",
            "ns1": "http://ns1.com"
        },
        "ns1***b": "value"
    }
}
```

Default	See examples above.
Presence	Optional However, if you specify <NamespaceBlockName>, you must also specify the other two elements.
Type	Strings

8.7.10.2.10 <Options>/<TextAlwaysAsProperty> <Options>/<TextNodeName> elements

Use these elements together.

If set to `true`, the content of the XML element is converted to a string property.

```
<TextAlwaysAsProperty>true</TextAlwaysAsProperty>
<TextNodeName>TEXT</TextNodeName>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

For the following XML:

```
<a>
  <b>value1</b>
  <c>value2<d>value3</d>value4</c>
</a>
```

If `TextAlwaysAsProperty` is set to true and `TextNodeName` specified as TEXT, the following JSON structure is generated:

```
{
  "a": {
    "b": {
      "TEXT": "value1"
    },
    "c": {
      "TEXT": [
        "value2",
        "value4"
        ],
        "d": {
          "TEXT": "value3"
        }
      }
    }
}
```

If `TextAlwaysAsProperty` is set to false and `TextNodeName` specified as TEXT, the following JSON structure is generated:

```
{
  "a": {
    "b": "value1",
    "c": {
      "TEXT": [
        "value2",
        "value4"
      ],
      {
        "d": "value3",
      }
    }
}
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Default	<TextAlwaysAsProperty>: false <TextNodeName>: N/A
Presence	Optional
Type	<TextAlwaysAsProperty>: Boolean <TextNodeName>: String

8.7.10.2.11 <Options>/<AttributeBlockName> <Options>/<AttributePrefix> elements

Use these elements together.

Lets you group values into a JSON block and append prefixes to the attribute names.

```
<AttributeBlockName>FOO_BLOCK</AttributeBlockName>
<AttributePrefix>BAR_</AttributePrefix>
```

Consider the following XML example:

```
<a attrib1="value1" attrib2="value2"/>
```

If both the attributes (AttributeBlockName and AttributePrefix) are specified as defined in the XML to JSON example, the following JSON structure is generated:

```
{
  "a": {
    "FOO_BLOCK": {
      "BAR_attrib1": "value1",
      "BAR_attrib2": "value2"
    }
  }
}
```

If only `AttributeBlockName` is specified, the following JSON structure is generated:

```
{
    "a": {
        "FOO_BLOCK": {
            "attrib1": "value1",
            "attrib2": "value2"
        }
    }
}
```

If only AttributePrefix is specified, the following JSON structure is generated:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
{
    "a": {
        "BAR_attrib1": "value1",
        "BAR_attrib2": "value2"
    }
}
If neither is specified, the following JSON structure is generated:
{
    "a": {
        "attrib1": "value1",
        "attrib2": "value2"
    }
}
```

Default	See examples above.
Presence	Optional
Type	String

8.7.10.2.12 <Options> / <OutputPrefix> <Options> / <OutputSuffix> elements

Use these elements together.

```
<OutputPrefix>PREFIX_</OutputPrefix>
<OutputSuffix>_SUFFIX</OutputSuffix>
```

Consider the following XML example:

```
<a>value</a>
```

If both the attributes (OutputPrefix and OutputSuffix) are specified as defined in the XML to JSON example, the following JSON structure is generated:

```
PREFIX_{
    "a": "value"
}_SUFFIX
```

If only `OutputPrefix` is specified, the following JSON structure is generated:

```
PREFIX_{
  "a" : "value"
}
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

If only `OutputSuffix` is specified, the following JSON structure is generated:

```
{
  "a" : "value"
}_SUFFIX
```

If neither `OutputPrefix` nor `OutputSuffix` is specified, the following JSON structure is generated:

```
{
    "a": "value"
}
```

Default	See samples above.
Presence	Optional
Type	String

8.7.10.2.13 XML to JSON Policy Error Codes

This section describes the error messages and flow variables that are set when this policy triggers an error. This information is important to know if you are developing fault rules for a proxy.

The XMLToJSON Policy type defines the following error codes:

Error name	HTTP status	Cause
EitherOptionOrFormat	500	See the fault string.
UnknownFormat	500	See the fault string.
FormatUnavailable	500	See the fault string.
SourceUnavailable	500	The variable specified in the <Source> element has to exist.
ExecutionFailed	500	See the fault string. Be sure the incoming message contains valid XML.
InvalidSourceType	500	See the fault string.
InCompatibleTypes	500	See the fault string.
OutputVariableIsNotAvailable	500	See the fault string.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Deployment errors

None.

Fault variables

These variables are set when a runtime error occurs.

Variables set	Where	Example
prefix].[policy_name].failed	The [prefix] is xmltojson. The [policy_name] is the name of the policy that threw the error.	xmltojson.XMLtoJSON-1.failed = true
fault.[error_name]	[error_name] = The specific error name to check for as listed in the table above.	fault.name Matches"Source Unavailable"

Example error response

Note:

For error handling, the best practice is to trap the errorcode part of the error response. Do not rely on the text in the faultstring, because it could change.

```
{
  "fault": {
    "faultstring": "XMLToJSON[XMLtoJSON-1]: Source xyz is not available",
    "detail": {
      "errorcode": "steps.xml2json.SourceUnavailable"
    }
  }
}
```

Example fault rule

```
<FaultRule name="XML to JSON Faults">
    <Step>
        <Name>AM-SourceUnavailableMessage</Name>
        <Condition>(fault.name Matches "SourceUnavailable") </Condition>
    </Step>
    <Step>
        <Name>AM-BadXML</Name>
        <Condition>(fault.name = "ExecutionFailed")</Condition>
    </Step>
    <Condition>(xmltojson.XMLtoJSON-1.failed = true) </Condition>
</FaultRule>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.11 JSON payloads in Assign Message and Raise Fault

When setting a JSON payload using an Assign Message or Raise Fault policy, users were sometimes required to use workarounds to ensure a JSON message was properly formatted at runtime, such as beginning the payload with a backslash "\" or specifying a variablePrefix and variableSuffix on the Payload element, even if no variables were used in the message.

With this enhancement, no workarounds are needed to ensure proper JSON message formatting, and variables can be specified using curly braces without creating invalid JSON. For example, the following inserts the value of message.content in the JSON message:

```
<Payload contentType="application/json">{"Message: " : "{message.content}"}</Payload>
```

If you used a workaround, your code will continue to work as is. You can also use variablePrefix and variableSuffix instead of curly braces to indicate variables.

8.7.12 Convert JSON to XML

SAP API Management enables developers to convert messages from the JavaScript object notation (JSON) format to the extensible markup language (XML) format by using the JSONtoXML policy type. This policy is useful for enabling backend XML services to support RESTful apps that require JSON (for example, due to a lack of an XML parsing capabilities on the client).

In a typical mediation scenario, a JSONtoXML policy on the inbound request flow is often paired with an XMLtoJSON policy on the outbound response flow. By combining policies this way, a JSON API can be exposed for services that natively support only XML.

For scenarios where APIs are consumed by diverse clients apps which may require either JSON or XML, the format of the response can be dynamically set by configuring JSONtoXML and XMLtoJSON policies to execute conditionally.

See Flow variables and conditions for an implementation of this scenario.

8.7.12.1 Sample

A set of configuration options are provided to customize the XML output by the JSONtoXML policy. The JSONtoXML policy defines a set of reasonable defaults. It is often useful to apply the default (empty) JSONtoXML policy and iteratively add configuration elements as required.

The minimum required configuration is the following:

```
<JSONToXML name="jsontoxml">
  <Options>
  </Options>
</JSONToXML>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.12.2 Configuring a JSON to XML Policy

All elements in this policy type are optional unless you need to define XML namespaces. IfNamespaceBlockName is set, DefaultNamespaceNodeName and NamespaceSeparator must also be specified.

The HTTP Content-type header of the source message must be set to application/json. If the HTTP Content-type header of the source message is not application/json, then the policy is not enforced.

The payload of the JSON message is parsed and converted into XML. The HTTP Content-type header of the XML-formatted message is set to application/xml.

> **Note**
>
> The name attribute for this policy is restricted to these characters: A-Z0-9._\-$ %. However, the Management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.

Field Name	Description
Source	The variable containing the JSON-formatted message to be converted to XML. Usually, you set this to request or response, depending on whether the message to be transformed is inbound or outbound. · If source is missing, it is treated as a simple message. For example,<Source>message</Source> · If the source variable cannot be resolved, or resolves to a non-message type, the policy execution fails.
OutputVariable	The variable name of the resultant XML-formatted message. Usually, you set this to request or response, depending on whether the message to be transformed is inbound or outbound. In most cases, an JSON request is transformed into an XML request. Similarly, a JSON response is usually transformed into an XML response. If OutputVariable is not specified, then the source variable is treated asOutputVariable. That is, the policy assumes that a JSON request, for example, is being converted into an XML request.

InvalidCharsReplacement	To assist with handling invalid XML that may cause issues with a parser, this setting replaces any JSON elements that produce invalid XML with the string. For example, the following setting:
	`<InvalidCharsReplacement>_</InvalidCharsReplacement>`
	Converts this JSON object
	`{`
	`"First%%%Name": "John"`
	`}`
	to this XML structure:
	`<First_Name>John<First_Name>`

TextNodeName	Converts a JSON property into an XML text node with the specified name. For example, the following setting: `<TextNodeName>age</TextNodeName>`

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name	Description

	converts this JSON:
	`{`
	`"person": {`
	`"firstName": "John",`
	`"lastName": "Smith",`
	`"age": 25`
	`}`
	`}`
	to this XML structure:
	`<person>`
	`<firstName>John</firstName>25<lastName>Smith</lastName>`
	`</person>`
	
	If TextNodeName is not specified, the XML is generated, using the default setting for a text node:
	`<person>`
	`<firstName>John</firstName>`
	`<age>25</age>`
	`<lastName>Smith</lastName>`
	`</person>`

NullValue	Indicates a null value. By default the value is NULL.
	For example the following setting:
	`<NullValue>I_AM_NULL</NullValue>`
	
	Converts the following JSON object:
	`{"person" : "I_AM_NULL"}`
	to the following XML element:
	`<person></person>`
	Where no value (or a value other than I_AM_NULL) is specified for the Null value, then the same payload converts to:
	`<person>I_AM_NULL</person>`

AttributeBlockName	Enables you to specify when JSON elements are converted into XML attributes (rather than XML elements).
	For example, the following setting converts properties inside an object named#attrs into XML attributes:
	`<AttributeBlockName>#attrs</AttributeBlockName>`
	The following JSON object:
	`{`
	`"person" : {`

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name	Description

	"#attrs" : {
	"firstName" : "John",
	"lastName" : "Smith"
	},
	"occupation" : "explorer",
	}
	}
	is converted to the following XML structure:
	<person firstName="John" lastName="Smith">
	<occupation>explorer</occupation>
	</person>

AttributePrefix	Converts the property starting with the specified prefix into XML attributes. Where the attribute prefix is set to @, for example:
	<AttributePrefix>@</AttributePrefix>
	Converts the following JSON object:
	{
	"person" : {
	"@firstName" : "John",
	"@lastName" : "Smith"
	"occupation" : "explorer",
	}
	}
	to the following XML structure:
	<person firstName="John" lastName="Smith">
	<occupation>explorer</occupation>
	</person>

NamespaceBlockName	JSON has no support for namespaces, while XML documents often require them. The NamespaceBlockName enables you to define a JSON property that serves as the source of a namespace definition in the XML that is produced by the policy. (This means that the source JSON must provide a property that can be mapped into a namespace that is expected by application that consumes the resulting XML.) For example, the following settings: <NamespaceBlockName>#namespaces</NamespaceBlockName> <DefaultNamespaceNodeName>$default</DefaultNamespaceNodeName> <NamespaceSeparator>:</NamespaceSeparator> Indicates that a property called #namespaces exists in the source JSON that contains at least one namespace designated as the default. For example: {

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Field Name	Description

	<pre>"population": {
	"#namespaces": {
	"$default": "http://www.w3.org/1999/people",
	"xmlns:exp": "http://www.w3.org/1999/explorers"
	},
	"person": "John Smith",
	"exp:person": "Pedro Cabral"
	}
	}</pre>converts to:<pre><person xmlns="http://www.w3.org/1999/people"
	xmlns:exp="http://www.w3.org/1999/explorers">
	<person>John Smith</person>
	<exp:person>Pedro Cabral</exp:person>
	</person></pre>

ArrayRootElementName	Converts a JSON array into a list of XML elements with specified parent and child element names. For example, the following settings: <ArrayRootElementName>Array</ArrayRootElementName> <ArrayItemElementName>Item</ArrayItemElementName> Converts the following JSON array: ["John Cabot", { "explorer": "Pedro Cabral" }, "John Smith"] into the following XML structure: <Array> <Item>John Cabot</Item> <Item> <explorer>Pedro Cabral</explorer> </Item> <Item>John Smith</Item> </Array>

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.7.12.3 Policy-specific Error Codes

The default format for error codes returned by Policies is:

```
{
  "code" : " {ErrorCode} ",
  "message" : " {Error message} ",
  "contexts" : [ ]
}
```

The JSONToXML Policy type defines the following error codes:

Error Code	Message
SourceUnavailable	JSONToXML[{0}]: Source {1} is not available
ExecutionFailed	JSONToXML[{0}]: Execution failed due to reason: {1}
OutputVariableIsNotAvailable	JSONToXML[{0}]: Output variable is not available.
InCompatibleTypes	JSONToXML[{0}]: String can not be assigned to message type.
InvalidSourceType	JSONToXML[{0}]: Invalid source type {0}. Validsource types are {1}.

8.8 Target Servers

API Platform Base Path: https:// managment_server_ip:8080>/v1/o/{org_name}

API Resource Path: `/environments/{env_name}/targetservers`

A named target (outbound) HTTP configuration, which can be used to configure load balancing for a single TargetEndpoint across multiple backend service URLs.

8.8.1 Create a TargetServer

Create a TargetServer in the specified environment. TargetServers are used to decouple TargetEndpoint HTTPTargetConnections from concrete URLs for backend services.

`environments/{env_name}/targetservers`

To do so, an HTTPConnection can be configured to use a LoadBalancer that lists one or more 'named' TargetSevers. Using TargetServers, you can create an HTTPTargetConnection that calls a different backend server based on the environment where the API proxy is deployed. See also Load Balancing Across Backend Servers.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Resource Summary	
Security	HTTP Basic
Content Type	application/json, text/xml
Category	TargetServers,

For example, instead of the following configuration:

```
<TargetEndpoint name="default">
  <HTTPTargetConnection>
    <URL>http://s1.mybackendservice.com</URL>
  </HTTPTargetConnection>
</TargetEndpoint>
```

```
You can reference a TargetServer as follows:
<TargetEndpoint name="default">
    <HTTPTargetConnection>
        <LoadBalancer>
            <Server name="target1" />
        </LoadBalancer>
  </HTTPTargetConnection>
</TargetEndpoint>
```

You can then set up a TargetServer called target1 in the 'test' environment that points to a test backend service, and a different TargetServer called target1 in the 'prod' environment that points to a production backend. When you 'promote' the API proxy with this configuration from the test to the prod environment, the TargetEndpoint will use the appropriate backend service, without requiring any changes in the API proxy configuration.

Note:

Characters you can use in the name are restricted to: A-Z0-9._\-$%

You can configure a TargetServer to use SSL.TargetServers expose the same SSL configuration settings as TargetEndpoints:

```
<TargetServer  name="TargetServer 1">
    <IsEnabled>true</IsEnabled>
    <Host>www.example.com</Host>
    <Port>443</Port>
    <SSLInfo>
        <Ciphers/>
        <ClientAuthEnabled>true</ClientAuthEnabled>
        <Enabled>true</Enabled>
        <IgnoreValidationErrors>false</IgnoreValidationErrors>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
        <KeyAlias>keystore-alias</KeyAlias>
        <KeyStore>keystore-name</KeyStore>
        <Protocols/>
        <TrustStore>truststore-name</TrustStore>
    </SSLInfo>
</TargetServer>
```

8.8.1.1 Resource URL

```
https://<managment_server_ip:8080>/v1/organizations/{org_name}/environments/{env_name}
/targetservers
```

8.8.1.2 Header Parameters

Name	Values	Description
Content-Type (required)	application/json	Specify the content type asapplication/json or text/xml.

8.8.1.3 Request Body

Name	Description	Default	Required?
Name	The name of the target server. You can choose anything you like.	N/A	Yes
Host	The DNS name or IP address of the machine that this TargetServer will refer to.	N/A	Yes
Port	The port on which the backend service is configured to listen. If you don't specify a port, it defaults to zero, which is invalid. If you're not sure, try port 80.	0	Yes
IsEnabled	A boolean (true/false) that determines whether	false	No

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Default	Required?
	this TargetServer is enabled or not. You can use this setting to take TargetServers out of rotation, without requiring you to delete the TargetServer definition.		
enabled	Enables one-way SSL. You must have defined a keystore containing the cert and private key.	false	No
clientAuthEnabled	Enables two-way, or client, SSL between SAP API Management Edge (server) and the app (client) making the request.	false	No
keyStore	The name of the keystore on SAP API Management Edge.	N/A	No
keyAlias	The alias specified when you uploaded the JAR file containing the cert and private key to the keystore.	N/A	No
trustStore	The name of the truststore on SAP API Management Edge that contains the certificate or certificate chain used for two-way SSL.	N/A	No
ignoreValidationErrors	If true, specifies to ignore SSL certificate errors. This is similar to the "-k" option to cURL.	false	No
ciphers	Specifies the ciphers supported by the virtual host. If no ciphers are specified, then all ciphers available for the JVM will be permitted.	All supported by the JVM	No
protocols	Specifies the protocols supported by the virtual	All supported by the JVM	No

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Name	Description	Default	Required?
	host. If no protocols are specified, then all protocols available for the JVM will be permitted.		

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.8.2 Delete a TargetServer

Delete a TargetServer configuration from an environment. Returns information about the deleted TargetServer.
`{environments}/env/targetservers/{targetserver_name}`

Resource Summary	
Security	HTTP Basic,
Content Type	application/json, text/xml
Category	TargetServers,

8.8.2.1 Resource URL

`https://<managment_server_ip:8080>/v1/organizations/{org_name}/environments/{env_name}/targetservers/{targetserver_name}`

8.8.3 List TargetServers in an environment

List all TargetServers in an environment.
`/environments/{env_name}/targetservers`

Resource Summary	
Security	HTTP Basic,
Content Type	application/json, text/xml
Category	TargetServers,

8.8.3.1 Resource URL

`https://<managment_server_ip:8080>/v1/organizations/{org_name}/environments/{env_name}/targetservers`

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.8.4 Get a TargetServer

Returns a TargetServer definition.

`/environments/{env_name}/targetservers/{targetserver_name}`

Resource Summary	
Security	HTTP Basic,
Content Type	application/json, text/xml
Category	TargetServers,

8.8.4.1 Resource URL

```
https://managment_server_ip:8080>/v1/organizations/{org_name}/environments/{env_name}/
targetservers/{targetserver_name}
```

8.8.5 Update a TargetServer

Modifies an existing TargetServer.

For example, use this method to toggles a TargetServer configuration from enabled to disabled. This is useful when TargetServers are used in load balancing configurations, and one or more TargetServers need to be taken out of rotation periodically. You could also use this API to modify the hostname of an enabled target server.

`/environments/{env_name}/targetservers/{targetserver_name}`

Resource Summary	
Security	HTTP Basic,
Content Type	application/json, text/xml
Category	TargetServers,

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.8.5.1 Resource URL

```
https://managment_server_ip:8080>/v1/organizations/{org_name}/environments/{env_name}/targetservers/{targetserver_name}
```

8.8.5.2 Header Parameters

Name	Values	Description
Content-Type (required)	application/json	Specify the content type asapplication/json or text/xml.

8.8.5.3 Request Body

Name	Description	Default	Required?
Name	The name of the target server. You can choose anything you like.	N/A	Yes
Host	The DNS name or IP address of the machine that this TargetServer will refer to.	N/A	Yes
Port	The port on which the backend service is configured to listen. If you don't specify a port, it defaults to zero, which is invalid. If you're not sure, try port 80.	0	Yes
IsEnabled	A boolean (true/false) that determines whether this TargetServer is enabled or not. You can use this setting to take TargetServers out of rotation, without requiring you to delete the TargetServer definition.	false	No
enabled	Enables one-way SSL. You must have defined a	false	No

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

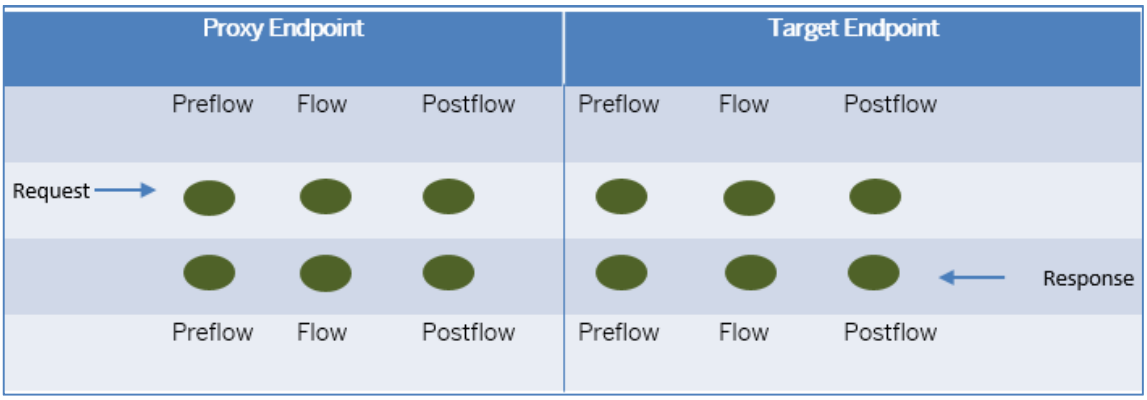
Name	Description	Default	Required?
	keystore containing the cert and private key.		
clientAuthEnabled	Enables two-way, or client, SSL between SAP API Management Edge (server) and the app (client) making the request.	false	No
keyStore	The name of the keystore on SAP API Management Edge.	N/A	No
keyAlias	The alias specified when you uploaded the JAR file containing the cert and private key to the keystore.	N/A	No
trustStore	The name of the truststore on SAP API Management Edge that contains the certificate or certificate chain used for two-way SSL.	N/A	No
ignoreValidationErrors	If true, specifies to ignore SSL certificate errors. This is similar to the "-k" option to cURL.	false	No
ciphers	Specifies the ciphers supported by the virtual host. If no ciphers are specified, then all ciphers available for the JVM will be permitted.	All supported by the JVM	No
protocols	Specifies the protocols supported by the virtual host. If no protocols are specified, then all protocols available for the JVM will be permitted.	All supported by the JVM	No

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.9 Java Callout Policy

The Java Callout Policy enables you to use Java to implement custom behavior that is not delivered out-of-the-box by SAP API Management policies.

The Java Callout Policy can be attached in the following locations:



8.9.1 Samples

```
<JavaCallout name="cityLookUp">
   <ClassName>com.sap.CityLookup</ClassName>
   <ResourceURL>java://CityLookup.jar</ResourceURL>
</JavaCallout>
```

8.9.2 Element reference

The element reference describes the elements and attributes of the JavaCallout policy.

```
<JavaCallout name="MyJavaCalloutPolicy">
   <ClassName>com.example.mypolicy.MyJavaCallout</ClassName>
   <ResourceURL>java://MyJavaCallout.jar</ResourceURL>
</JavaCallout>
```

8.9.3 <JavaCallout> attributes

```
<JavaCallout name="MyJavaCalloutPolicy">
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Attribute	Description	Default	Presence
name	The internal name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the SAP API Management Edge UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric. Optionally, use the <DisplayName> element to label the policy in the management UI proxy editor with a different, natural-language name.	N/A	Required
continueOnError	Set to false to return an error when a policy fails. This is expected behavior for most policies. Set to true to have flow execution continue even after a policy fails.	false	Optional
enabled	Set to true to enforce the policy. Set to false to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	true	Optional
async	This attribute is deprecated.	false	Deprecated

8.9.4 <DisplayName> element

Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

```
<DisplayName>Policy Display Name</DisplayName>
```

Default: N/A
If you omit this element, the value of the policy's nameattribute is used.

Presence: Optional

Type: String

8.9.5 <ResourceURL> element

Specifies the name and location of the main JAR file used to execute the callout. This JAR file contains compiled Java code written to the Java API exposed by the API Platform processing pipeline.

```
<JavaCallout name="MyJavaCalloutPolicy">
   <ResourceURL>java://MyJavaCallout.jar</ResourceURL>
   <ClassName>com.example.mypolicy.MyJavaCallout</ClassName>
</JavaCallout>
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

- Default: N/A
- Presence: Required
- Type: String

8.9.6 <ClassName> element

Specifies the name of the Java class that executes the callout.

```
<JavaCallout name="MyJavaCalloutPolicy">
   <ResourceURL>java://MyJavaCallout.jar</ResourceURL>
   <ClassName>com.example.mypolicy.MyJavaCallout</ClassName>
</JavaCallout>
```

- Default: N/A
- Presence: Required
- Type: String

8.9.7 Error reference

This section describes the error messages and flow variables that are set when this policy triggers an error. This information is important to know if you are developing fault rules to handle errors.

8.9.7.1 Error code prefix

`steps.messagelogging`

8.9.7.2 Runtime errors

These errors can occur when the policy executes.

The JavaCallout policy type defines the following error codes:

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Error name	Fault string	HTTP status	Occurs when
ExecutionError	Failed to execute JavaCallout.[policy_name]	500	See fault string.

8.9.7.3 Fault variables

These variables are set when this policy triggers an error.

Variables set	Where	Example
[prefix].[policy_name].failed	[prefix]: javacallout [policy_name]: The name of the policy to check.	javacallout.JC-GetUserData.failed =true
fault.[error_name]	[error_name] = The specific error name to check for as listed in the table above.	fault.name Matches "ExecutionError"

8.9.7.4 Example error response

```
{
   "fault":{
      "faultstring":"Failed to execute JavaCallout. [policy_name]",
      "detail":{
         "errorcode":"javacallout.ExecutionError"
      }
   }
}
```

8.9.7.5 Example fault rule

```
<FaultRule name="JavaCalloutFailed">
    <Step>
        <Name>AM-JavaCalloutError</Name>
    </Step>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
    <Condition>(fault.name Matches "ExecutionError") </Condition>
</FaultRule>
```

8.9.8 Referencing the Java class in a JavaCallout policy

Before we take a look at the Java implementation, let's see how to reference a Java class (or classes) in a policy. In this cookbook sample, there is a policy called `cityLookup.xml`. You can find it in the standard policies directory inside the apiproxy folder. If you downloaded the code, you can find the policy XML here: `doc-samples/java-cookbook/apiproxy/policies/cityLookup.xml`.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<JavaCallout name="cityLookUp">
 <ClassName>com.sap.CityLookup</ClassName>
 <ResourceURL>java://CityLookup.jar</ResourceURL>
</JavaCallout>
```

This is a very simple, straightforward policy! Besides a name (cityLookup), this policy references the main Java class file and a JAR file. As you might expect, the class file and all supporting classes are stored in the JAR. If you refer back to the flow, you will note that the cityLookup policy is called in the `<Request>` stage of the ProxyEndpoint. This means that the Java class is executed at that stage in the flow.

Let's look at the Java class next, and then at how it is compiled, packaged, and deployed.

8.9.9 Coding the Java class

This example Java class is deliberately simple and trivial. However, it illustrates some fundamental concepts common to any Java code that is executed through a JavaCallout policy.

```
package com.sap;

import com.sap.flow.execution.ExecutionContext;
import com.sap.flow.execution.ExecutionResult;
import com.sap.flow.execution.spi.Execution;
import com.sap.flow.message.MessageContext;

public class CityLookup implements Execution{

     public ExecutionResult execute(MessageContext messageContext,
        ExecutionContext executionContext) {
    try {
         int w = 0;
         String cityName =
```


© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
                messageContext.getMessage().getQueryParam("city").toUpperCase();

        if (cityName.equals("MILAN")) w=718345;
        else if (cityName.equals("ROME")) w=721943;
        else if (cityName.equals("VENICE")) w=725746;

        if(w>0) {
             messageContext.getRequestMessage().setQueryParam("w", w);
        } else {
             //default to palo alto
             messageContext.getRequestMessage().setQueryParam("w", 2467861);
        }

        return ExecutionResult.SUCCESS;

       } catch (Exception e) {
        return ExecutionResult.ABORT;
       }
    }
}
```

The most important things to remember about this Java implementation are:

- Imports classes from the com.sap.flow.execution and com.sap.flow.message packages. These packages must be included in the JAR file that is packaged and deployed. We'll cover packaging and deployment later in this topic. In the cookbook sample download, you can find these packages in doc-samples/java-cookbook/lib.
- Implements Execution. Any Java code that is executed within an API proxy must implement Execution. <<note about javadoc>>

8.9.10 Error handling

The execute() method in our Java class throws an exception if it can't recognize the city query parameter (only three Italian city names are valid: Milan, Venice, and Rome). On error, the method returns an ExectionResult.ABORT status. To handle this error situation, we added a RaiseFault policy to the flow. If you refer back to the proxy flow shown previously, you will see that this policy, called Fault.MissingCityName, is added to the policy's <FaultRules> element.

Here is the policy XML. If you downloaded the sample code from Github, you can find this policy XML here: doc-samples/java-cookbook/apiproxy/policies/Fault.MissingCityName.xml.

```
<RaiseFault name="Fault.MissingCityName">
 <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
 <FaultResponse>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

```
 <Set>
 <Headers/>
 <Payload contentType="application/xml">
 <error>
 <httpResponseCode>409</httpResponseCode>
 <errorCode>error_409</errorCode>
 <errorMessage>Missing City Name in Query Parameter</errorMessage>
 <errorMoreInfo>Valid cities are venice, milan, and rome. </errorMoreInfo>
 </error>
 </Payload>
 <StatusCode>409</StatusCode>
 <ReasonPhrase>Conflict</ReasonPhrase>
 </Set>
 </FaultResponse>
</RaiseFault>
```

The policy simply sets an error code and provides information about the error. This information is sent to the client. You can read more about the structure of RaiseFault policies in Raise Fault policy.

8.9.11 Specifying the default target endpoint

Our final task is to specify the Yahoo Weather API URL on the proxy's TargetEndPoint. If you downloaded the code, you can find the policy XML here: doc-samples/java-cookbook/apiproxy/targets/default.xml. Here is the source code:

```
<TargetEndpoint name="default">
 <HTTPTargetConnection>
 <Properties/>
 <URL>http://weather.yahooapis.com/</URL>
 </HTTPTargetConnection>
</TargetEndpoint>
```

8.9.12 Testing the example

Before you can deploy and test this sample, you must first compile the CityLookup.java class and generate a JAR file that contains that class plus the required packages that the custom class requires. The required libraries are expressions-1.0.0.jar and message-flow-1.0.0.jar, and they are located in the doc-samples/java-cookbook/lib folder.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

If you have not already done so, try to download, deploy, and run the java-cookbook sample, which you can find in the doc-samples folder in the doc samples repositor. Just follow the instructions in the README file in the java-cookbook folder.

To summarize, you can call the API as follows. Substitute {myorg} with your organization name. This example assumes you are using the test environment.

```
curl http://{hostname}:{port}/java-cookbook/forecastrss?city=venice
```

The response includes weather information for Venice in XML format. Here are the first few lines of the response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<rss version="2.0" xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">
<channel>

<title>Yahoo! Weather - Venice, IT</title>
<link>http://us.rd.yahoo.com/dailynews/rss/weather/Venice__IT/*http://weather.yahoo
.com/forecast/ITXX0085_f.html</link>
<description>Yahoo! Weather for Venice, IT</description>
<language>en-us</language>
<lastBuildDate>Fri, 06 Dec 2013 3:54 pm CET</lastBuildDate>
<ttl>60</ttl>
<yweather:location city="Venice" region="VN" country="Italy"/>
<yweather:units temperature="F" distance="mi" pressure="in" speed="mph"/>
<yweather:wind chill="46" direction="" speed="" />
<yweather:atmosphere humidity="76" visibility="3.73" pressure="29.85" rising="0" />
<yweather:astronomy sunrise="7:35 am" sunset="4:26 pm"/>
...
```

Remember that the Java class is hard-wired to recognize only Venice, Milan, and Rome. Try invoking the API with a city that is not in this list, like Sacremento.

```
curl http://{hostname}:{port}/java-cookbook/forecastrss?city=sacramento
```

This call results in an error and the JavaCallout policy follows it's Fault path. The resulting message returned is generated by the RaiseFault policy, discussed previously:

```
<error>
 <httpResponseCode>409</httpResponseCode>
 <errorCode>error_409</errorCode>
 <errorMessage>Missing City Name</errorMessage>
 <errorMoreInfo>Provide a city parameter in the querystring e.g.
city=venice</errorMoreInfo>
</error>
```


CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.9.13 Usage notes

- For lightweight operations, such as API calls to remote services, we recommend using the ServiceCallout policy. See Service Callout policy.
- For relatively simple interactions with message content, such as modifying or extracting HTTP headers, parameters, or message content, you can use JavaScript or Python languages.
- Place the JAR in an API proxy under /resources/java. If your JavaCallout relies on additional third-party libraries packaged as independent JAR files, then place those JAR files in the /resources/java directory as well to ensure that they are loaded correctly at runtime. If you are using the management UI to create or modify the proxy, add a new resource and specify an additional dependent JAR file. If there are multiple JARs, simply add them as additional resources. You do not need to modify the policy configuration to refer to additional JAR files. Putting them in /resources/java is sufficient.
- System calls, for example network I/O, filesystem read/writes, current user info, process list, and CPU/memory utilization are not permitted by the security model. Although some such calls may be functional, they are unsupported and liable to be actively disabled at any time. For forward compatibility, you should avoid making such calls in your JavaCallout.

8.10 Flow Callout Policy

Use the Flow Callout policy to call out to a shared flow from either an API proxy or another shared flow.

Note: The ability to create shared flows is available only on request. The feature will not appear in the user interface unless activated for your org.

In a shared flow, you create a sequence of steps that you can reuse at run time from multiple places. These steps are implemented as policies, as with an API proxy. The Flow Callout policy gives you a way to call to the shared flow from API proxies and other shared flows.

- For example, imagine that you've implemented a shared flow with security features such as API key verification and OAuth token validation. Using the Flow Callout policies, you can route requests through that flow from multiple API proxies. In this way, you can enforce standard security in one place, then use it from multiple others.
- You can call one shared flow from another by implementing a Flow Callout policy in a shared flow. For example, the security shared flow could use a Flow Callout policy to call to another shared flow that performs a transformation on the incoming request message

For more on shared flows, see Reusable shared flows. You can also call a shared flow from outside API proxy code to do request pre-processing or response post-processing.

This policy can be attached in the following locations.

	ProxyEndpoint			TargetEndpoint			
	PreFlow	Flow	PostFlow	PreFlow	Flow	PostFlow	
Request →	●	●	●	●	●	●	
	●	●	●	●	●	●	*← Response*
	PostFlow	Flow	PreFlow	PostFlow	Flow	PreFlow	

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Samples

This example creates a callout to a local API proxy (that is, one in the same organization and environment) called data-manager, specifying its proxy endpoint whose name is default.

```
<LocalTargetConnection>
  <APIProxy>data-manager</APIProxy>
  <ProxyEndpoint>default</ProxyEndpoint>
</LocalTargetConnection>
```

8.10.1 About the Flow Callout Policy

There are many scenarios where you can use a Service Callout policy in your API proxy. For example, you can configure an API proxy to make calls to an external service to deliver geolocation data, customer reviews, items from a partner's retail catalog, and so on.

8.10.2 Element reference

Following are elements and attributes you can configure on this policy:

```
<FlowCallout async="false" continueOnError="false" enabled="true" name="Flow-Callout-1">
    <DisplayName>Custom label used in UI</DisplayName>
    <SharedFlowBundle>thereferencedsharedflowbundle</SharedFlowBundle>
</FlowCallout>
```

8.10.3 <FlowCallout> attributes

```
<ServiceCallout async="false" continueOnError="false" enabled="true" name="Service-Callout-1">
```

The following attributes are common to all policy parent elements.

Attribute	Description	Default	Presence
Name	The internal name of the policy. Characters you can use in the name are restricted to: A-Z0-9._\-$ %. However, the Edge management UI enforces additional restrictions, such as automatically removing characters that are not alphanumeric.	N/A	Required

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Attribute	Description	Default	Presence
	Optionally, use the <DisplayName> element to label the policy in the management UI proxy editor with a different, natural-language name.		
continueOnError	Set to false to return an error when a policy fails. This is expected behavior for most policies. Set to true to have flow execution continue even after a policy fails.	False	Optional
enabled	Set to true to enforce the policy. Set to false to "turn off" the policy. The policy will not be enforced even if it remains attached to a flow.	True	Optional
async	This attribute is deprecated.	False	Deprecated

8.10.4 <DisplayName> element

Use in addition to the name attribute to label the policy in the management UI proxy editor with a different, natural-language name.

```
<DisplayName>Policy Display Name</DisplayName>
```

Default: N/A

If you omit this element, the value of the policy's name attribute is used.

Presence: Optional

Type: String

8.10.5 <SharedFlowBundle> element

Specifies the name of the shared flow to call. This element's value should be the same as the value of the target SharedFlowBundle element's name attribute.

```
<SharedFlowBundle/>
```

In the simplest example, you give the name of the shared flow being called as a value for this element. That is, this element's value must be the same as the shared flow's name attribute value.

```
<SharedFlowBundle>Shared-Flow-Name</SharedFlowBundle>
```

Default: N/A

Presence: Required

Type: String

Attributes: None

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

8.10.6 Flow Variables

Flow variables enable dynamic behavior of policies and flows at runtime, based on HTTP headers, message content, or flow context.

Attribute	Description
apigee.edge.sharedflow.name	Scope: During the shared flow execution Type: String Permission: Read The value of the shared flow's name attribute
apigee.edge.flowhook.name	Scope: During execution of the shared flow attached to the flow hook. Type: String Permission: Read The name of the flow hook.

8.10.7 Error Codes

None.

CUSTOMER
© 2014 SAP SE or an SAP affiliate company. All rights reserved.

9 Glossary

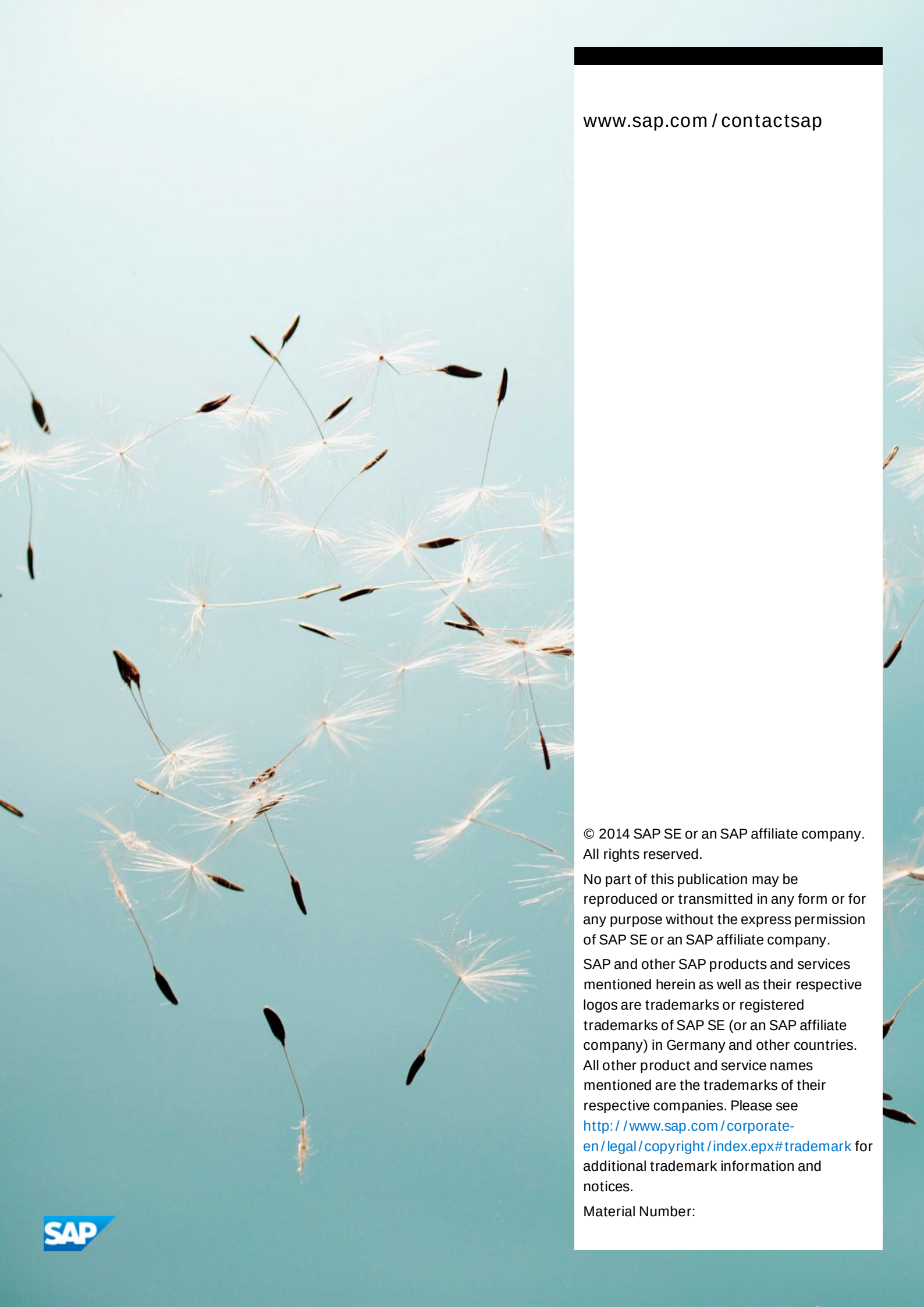
Term	Definition
API	An "Application Programming Interface" is a technology architecture that makes it easy for one application to consume capabilities or data from another application. APIs enable developers to easily access and reuse application logic built by other developers.
API Proxy	A type of application that runs on SAP API Management and exposes a facade for one or more APIs, generic HTTP services. An API proxy is implemented as a set of configuration files, policies, and code that rely on a set of resources provided by SAP API Management Services. API proxies can be generated and configured using the SAP API Management UI, or they can be implemented locally in a text editor or IDE.
API base path and resources	APIs defined by network addresses and URIs. An API is made up of a "base path" and a set of "API resources". Every API proxy defines a base path and, optionally, multiple API resource paths. You can think of an API simply as a set of URIs, all of which share a common base path.
API Product	A collection of API resources (URIs) combined with a quota, or 'service plan', which is published to app developers at design time. API products can in turn be bundled into API packages. An API key is bound to one or more API products, enforcing a binding between an app and the bundle of URIs that the app is permitted to consume.
API package	A collection of API products that are presented to developers as a bundle, and typically associated with a rate plan.
Apps	· Your developers use apps to access the resources in your API products. When you create an app, you select the API product to include, and SAP API Management generates a key. Each app has a single key that provides access to multiple API products. Apps allow you to control who can access your resources. You can control who has access to your API products by revoking and refreshing an app's key. And you can control access to bundles of resources by revoking or deleting access to the products in an app.
Environment	A runtime execution context for API proxies. An API proxy must be deployed to an environment before the API it exposes is accessible over the network. By default, organizations are provisioned with two environments: 'test' and 'prod'. · The 'test' environment is typically used for deploying API proxies during development. The 'prod' environment is typically used for promoting API proxies from the test environment after they have been fully developed and tested.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

Term	Definition
Organization	A container for all the objects in the SAP API Management account, including API proxies, API products, API packages, apps, and developers. A user account is required for each organization for which you are a member. (Most users will have an account in only one organization.) You need to supply your credentials (username and password) and the name of your organization with each API request you submit.
Policy	A processing step that executes as an atomic, reusable unit of logic within an API proxy processing flow. Typical policy-based functionality includes transforming message formats, enforcing access control, calling remote services for additional information, masking sensitive data from external users, examining message content for potential threats, caching common responses to improve performance, and so on. Policies may be conditionally executed based on the content or context of a request or response message. For example, a transformation policy may be executed to customize a response format if the request message was sent from a smartphone.
resource path	A RESTful concept, a resource path is a uniform resource identified (URI) that identifies the network path to a given resource.
version	The version of the developer-facing API interface. For example,pivotaltracker.com/services/v3 (This term is distinguished from 'revision', which is the numbered, version-controlled package of configuration and policies bundled into an API Proxy. In short, API interfaces have versions, while API Proxies have revisions.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.



www.sap.com/contactsap

© 2014 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies. Please see http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark for additional trademark information and notices.

Material Number: