Public API (PAPI) Web Services Guide

SAP Manufacturing Execution 15.1

Target Audience

- Developers
- Technology consultants

Document Version 1.0 – October 2015

THE BEST-RUN Bbusinesses RUN SAP

SAP SE
Dietmar-Hopp-Allee 16
69190 Walldorf
Germany
T +49/18 05/34 34 34
F +49/18 05/34 34 20
www.sap.com

Documentation in the SAP Service Marketplace
You can find this documentation at the following Internet address: http://service.sap.com/instguides

Typographic Conventions

Type Style	Represents
Example Text	Words or characters that appear on the screen. These include field names, screen titles, pushbuttons as well as menu names, paths and options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, titles of graphics and tables
EXAMPLE TEXT	Names of elements in the system. These include report names, program names, transaction codes, table names, and individual key words of a programming language, when surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Screen output. This includes file and directory names and their paths, messages, names of variables and parameters, source code as well as names of installation, upgrade and database tools.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Pointed brackets indicate that you replace these words and characters with appropriate entries.
`EXAMPLE TEXT`	Keys on the keyboard, for example, function keys (such as `F2`) or the ENTER key.

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

History of Changes

The following table provides an overview of the most important changes that were made in the latest versions.

Version	Important Changes
1.0	Initial version.

Table of Contents

Introduction

PAPI (public API) web services expose the SAP ME public APIs to provide the web service equivalent of the SAP ME PAPI services. Each of the PAPI services consists of one or more operations (methods) which may be used to invoke business functions within SAP ME. These operations are useful for implementing font-end UIs, integrating with manufacturing equipment and business systems, or extending the SAP ME solution to meet the needs of your organization. This document provides you with the information you need to get started with the PAPI web services.

Prerequisites

In order to utilize the PAPI web services, your environment must meet or exceed the following prerequisites:

- You have installed SAP NetWeaver 7.5 with the following usage types:
  - *Application Server Java*
  - *Adobe Document Services (ADS)*
- You have installed SAP ME 15.1

To proceed with the source code examples, you need the following:

- soapUI or equivalend web services tool
- Microsoft Visual Studio 2008 (SP1)
- SAP NetWeaver Development Studio (NWDS) version 7.3 or later, or the SAP ME SDK 15.0 Eclipse IDE
- Understanding of the W3C Web Services standard (such as SOAP and WSDL)

References

For more information, see the following topics:

- Preparing Communication Profiles in SAP NetWeaver 7.5 documentation.
- Configuring Web Services Exposed by Applications in SAP NetWeaver 7.5 documentation.
- SAP Plant Connectivity in SAP Plant Connectivity (PCo) 15.1 documenation.
- Security Considerations for Service Groups in SAP NetWeaver 7.5 documentation.

Standards Compliance

The PAPI web services conform to the *Java Architecture for XML Binding* (JAXB) 2.0 specification. In addition, the PAPI web services include support for the *Java API for XML-Based Web* Services *(JAX-WS)* 2.1 specification.

Getting Started

This section describes what is needed to effectively utilize the PAPI web services and how to send a PAPI web service request through soapUI (a tool used for sending/testing web service requests).

You may also find the *Application Development* section helpful if you intend to create your own client applications that utilize the PAPI web services.

There are several web services available which comprise the vast majority of functions you might need to perform within SAP ME. This section will assist you by providing suggestions for when to use the PAPI web services, configuration of the services, and how to get started using the PAPI web services.

RECOMMENDATION

Review all the information in this document before you begin developing your applications. Doing so will assist you in making effective use of the services available to you.

When to Use the PAPI Web Services

The PAPI web services provide developers with the capability to perform SAP ME API functions.

There are over 150 individual services within the PAPI web service package, each consisting of one or more operations. These services may be used to perform various business functions within SAP ME and may be invoked directly from your applications.

For more information, see *SAP ME 15.1 Javadoc*.

The benefits of using the PAPI web services include:

- Client applications can use a programming language or platform that supports the SOAP protocol (such as C# or Java)
  Caution: For case-insensitive programming languages such as Visual Basic .NET, you must manually rename classes generated by the IDE from PAPI WSDL.
- Ideal for testing scenarios.
- Enable in-house project teams to build new solutions that interface to SAP ME from an alternative technology platform.

Consider the following to assist you in determining when to use the PAPI services. You organization has:

- A Front-end application that users are familiar with and would like to use it to interact with SAP ME.
- A very complex machine interface that performs several unique actions that would be challenging to implement with SAP PCo (see SAP Plant Connectivity for additional options).
- An existing system which requires the data to be exchanged with SAP ME (either in real-time or on a scheduled task).

The above are just some suggestions for possible reasons for using the PAPI web services.

Development Platforms

The PAPI web services are standards compliant web services that allow them to be accessed with various technology frameworks and platforms. The environment you choose to develop with primarily depends on the following:

- Development environment must support the SOAP protocol.
- Development environment used may be dependent upon your organizations policy for Software Development standards.

Most development environments provide developers with tools to reference a WSDL and import the structure into their solutions. Such solutions include, but are not limited to .NET Visual Studio, NWDS, and Eclipse (JEE).

API Support Policy

The PAPI web services are supported by SAP. Should you require assistance, consult your available support options for direction on how to obtain Technical support.

SAP ME Javadoc Navigation

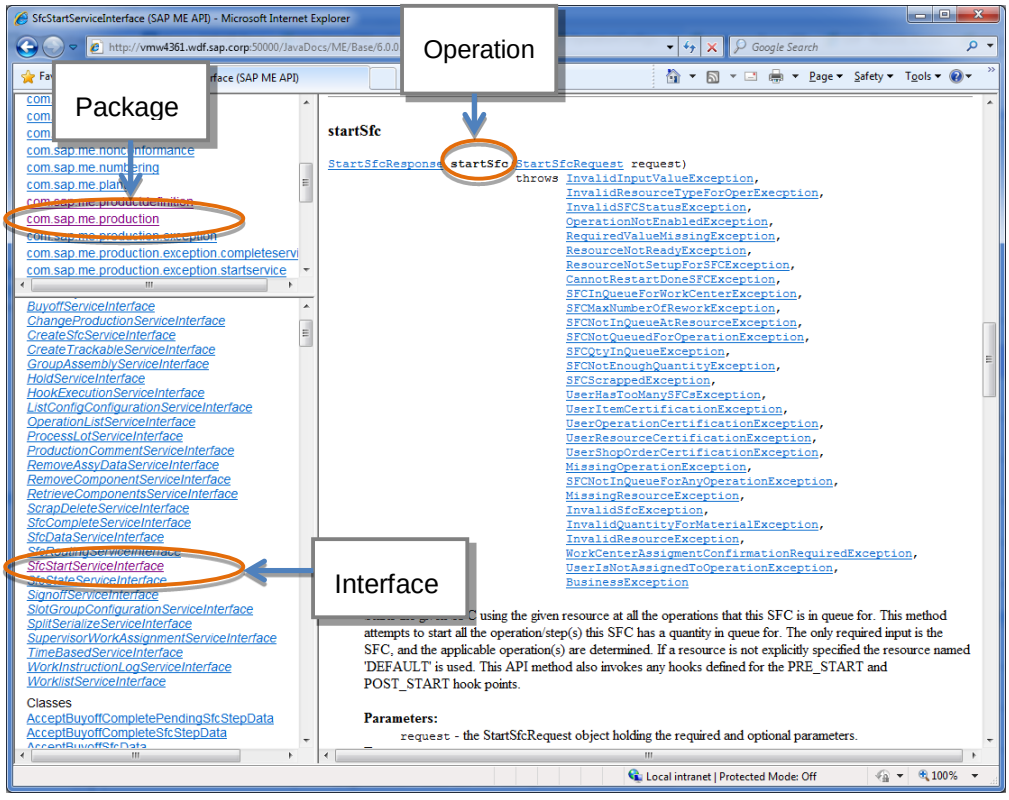
For detailed documentation of each of the PAPI web services and their associated operations, see the associated Javadoc that were included with your release of SAP ME. Each version of SAP ME shipped includes the Javadoc.

The SAP ME Javadoc is an important reference for describing the operations and determining the requirements for web service calls.

The Javadocs are organized such that the upper left-hand panel identifies the packages that are available. By clicking a specific package you can see the Interfaces that are associated with that package at the bottom left panel. After clicking on the interface you wish to view, you will then see the method summary on the right-hand panel. This panel provides all available operations for the interface. Most all of the interfaces identified exist as PAPI services.

For each operation you will see one or more parameters required to complete the request. In order to send the request to SAP ME, you must populate the request with the appropriate object requirements. In the example below, you will see a *startSfc* request which requires the *StartSfcRequest* to be populated.

Each operation includes a brief description of the business function it supports. This description does not include a detailed breakdown of all the business logic contained within the operation. Consult SAP ME 15.1 application help and functional training materials to understand the configuration and business rules within the application.



Clicking the link for the `StartSfcRequest` object will provide you with the details needed to craft the required object. For instance, the `StartSfcRequest` requires the `sfcRef` to be populated within the request.

com.sap.me.production

Class StartSfcRequest

java.lang.Object
  └ com.sap.me.production.StartSfcRequest

All Implemented Interfaces:
Serializable

```
public class StartSfcRequest
extends Object
implements Serializable
```

The schema/DTO containing all the parameters associated with the request to start an SFC.

Java class for StartSfcRequest complex type.

The following schema fragment specifies the expected content contained within this class.

```
<complexType name="StartSfcRequest">
  <complexContent>
    <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
      <sequence>
        <element name="sfcRef" type="{http://www.sap.com/me/common}Handle"/>
        <element name="itemRef" type="{http://www.sap.com/me/common}Handle" minOccurs="0"/>
        <element name="operationRef" type="{http://www.sap.com/me/common}Handle" minOccurs="0"/>
        <element name="resourceRef" type="{http://www.sap.com/me/common}Handle" minOccurs="0"/>
        <element name="userRef" type="{http://www.sap.com/me/common}Handle" minOccurs="0"/>
        <element name="workCenterRef" type="{http://www.sap.com/me/common}Handle" minOccurs="0"/>
        <element name="qty" type="{http://www.sap.com/me/common}Qty" minOccurs="0"/>
        <element name="rework" type="{http://www.sap.com/me/common}Boolean" minOccurs="0"/>
        <element name="dateTime" type="{http://www.sap.com/me/common}DateTime" minOccurs="0"/>
        <element name="noSfcValidation" type="{http://www.sap.com/me/common}Boolean" minOccurs="0
        <element name="confirmed" type="{http://www.sap.com/me/common}Boolean" minOccurs="0"/>
        <element name="checkWorkCenterAssigment" type="{http://www.sap.com/me/common}Boolean" min
        <element name="workStationRef" type="{http://www.sap.com/me/common}Handle" minOccurs="0"/
      </sequence>
    </restriction>
  </complexContent>
</complexType>
```

Quick Start

This section will assist you in sending your initial PAPI web service request to SAP ME.

If the PAPI web services have already been installed and configured at your site, you may skip the Installation and Configuration sections. However, if you are not yet familiar of how to locate the WSDLs required for your application, see Choosing the WSDL. This section helps you identify the appropriate WSDL and the URL associated with the WSDL from within the SAP NetWeaver Administrator.

Installation

Before continuing with this section, verify that your environment meets the recommended prerequisites. If you, or someone else in your organization, have already installed and configured the PAPI web services on your SAP ME server, then you may skip this section.

The PAPI web services are installed as part of the standard SAP ME product. Therefore, deploying SAP ME, PAPI services should be available to you.

Configuration

This section will assist you in configuring the PAPI web services.

There are over 150 PAPI web services available for use. For more information, see *SAP ME Javadoc*.

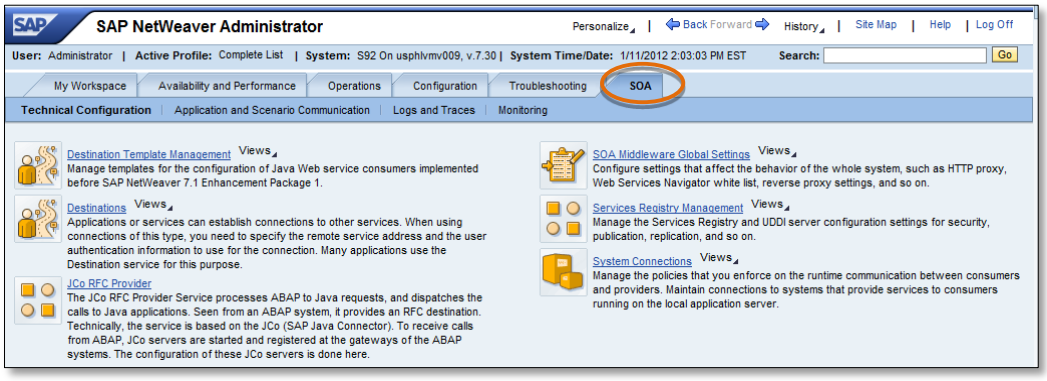
Configuring each service individually would take an extensive amount of time. To simplify the assignment of Authentication and Configuration information to the PAPI web services, we recommend that you create a *Communication Profile*. The use of a *Communication Profile* both simplifies and expedites the configuration process.

The *Communication Profile* is used to directly access the WSDL using the Authentication, Reliable Messaging and Transport binding settings associated with the profile. It is possible to configure the settings on a per-service basis, however most organizations will find using a Communication Profile to require less system administration. For that reason, this section focuses primarily on the Communication Profile methodology.

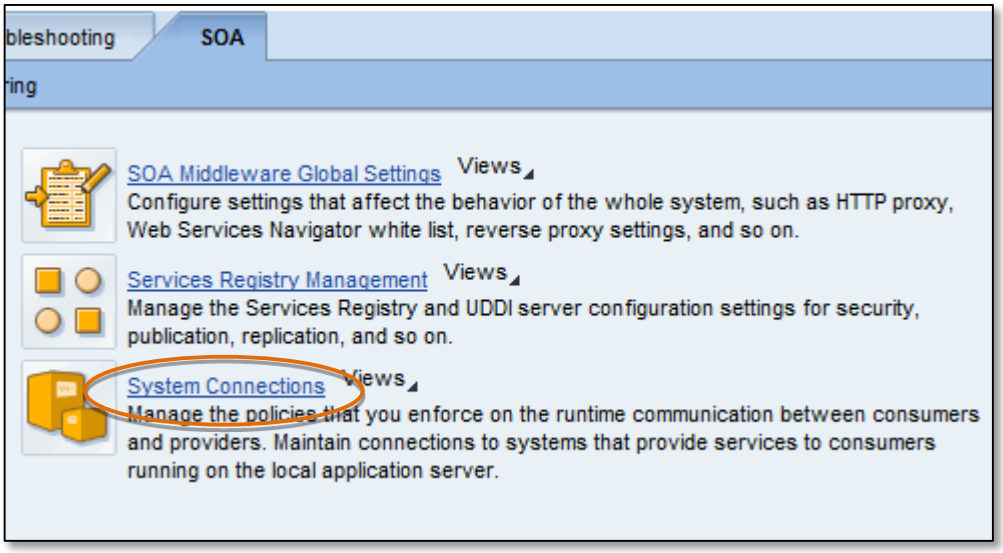
Creating a Communication Profile

The first step in the configuration process is creating the Communication Profile. This section will guide you through the process.

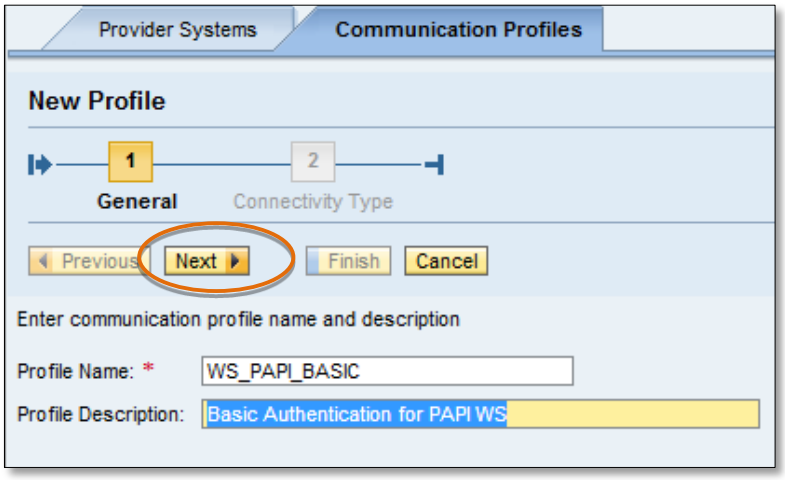
1. Open Internet Explorer and enter the URL: `http://<server>:<port>/nwa` (specifying your server/host and port number).
2. Log on to SAP NetWeaver Administrator by entering your *User* and *Password* and choosing the *Log On* button.
3. Choose the *SOA* tab.



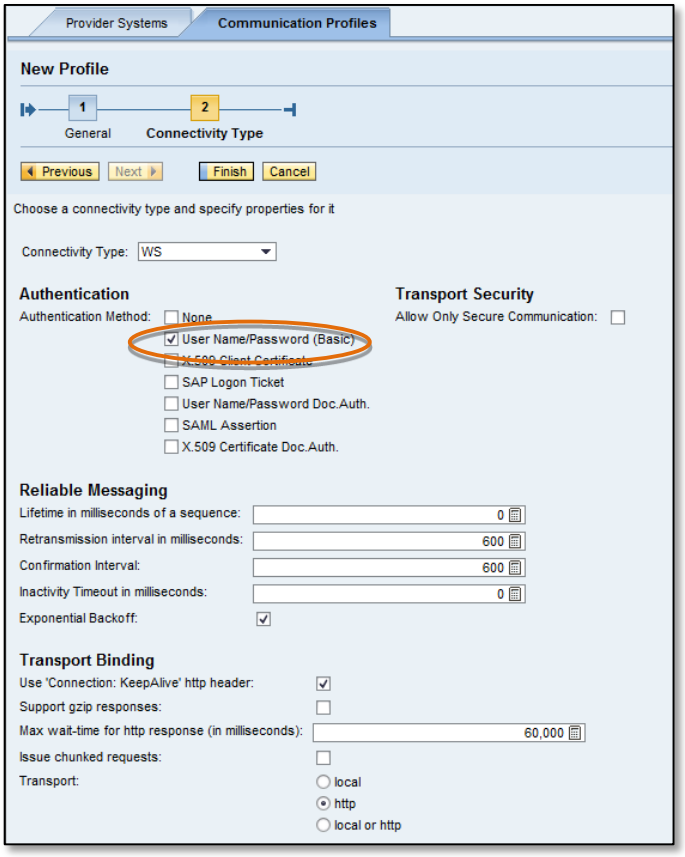
4. On the *SOA* tab page, click the *System Connections* link on the lower-right.



5. Choose the *Communication Profiles* tab.
   Various pre-configured profiles appear.
   We will create a new profile to be used exclusively for the PAPI web services using *Basic Authentication* (requiring a user name and password).
6. On the *Communication Profiles* tab page, choose *New*.
7. Enter a *Profile Name* and *Profile Description* for the *Communication Profile* and then choose the *Next* button.
   In the example below, we have entered a profile name of `WS_PAPI_BASIC` and a profile description of *Basic Authentication for PAPI WS*..



8. The *Connectivity Type* should default to *WS* (or web services). If it is not set properly, select *WS*.
9. For the *Authentication Method,* deselect *None* and select *User Name/Password (Basic)*.
   All other settings may remain with their default values.



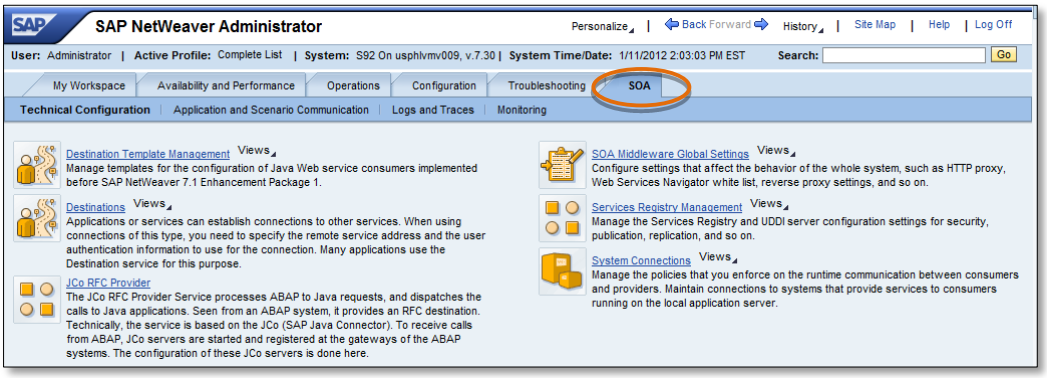
10. Choose the *Finish* button.
    A list of *Communication Profiles* now includes the recently added `WS_PAPI_BASIC` profile name.

Applying the Communication Profile

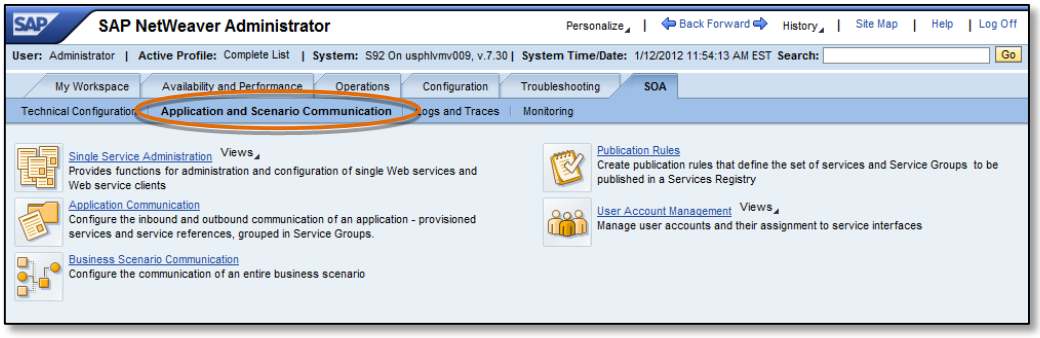
After creating a *Communication Profile*, you need to assign it to the PAPI web services. This section walks you through the process of applying the *Communication Profile* to all PAPI web services. This methodology provides a quick and easy means of changing the communication settings for all PAPI web services.

Before proceeding, you must have already created a *Communication Profile* as described in the previous section.

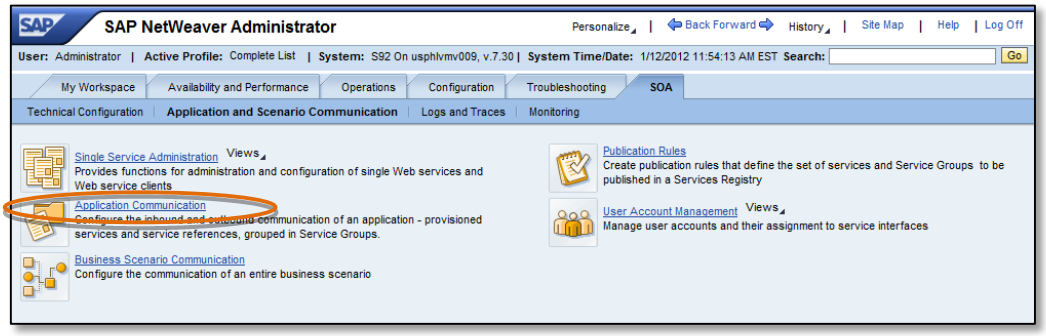
1. In SAP NetWeaver Administrator, choose the *SOA* tab.



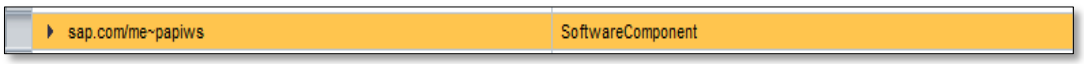
2. On the *SOA* tab page, click the *Application and Scenario Communication* link.



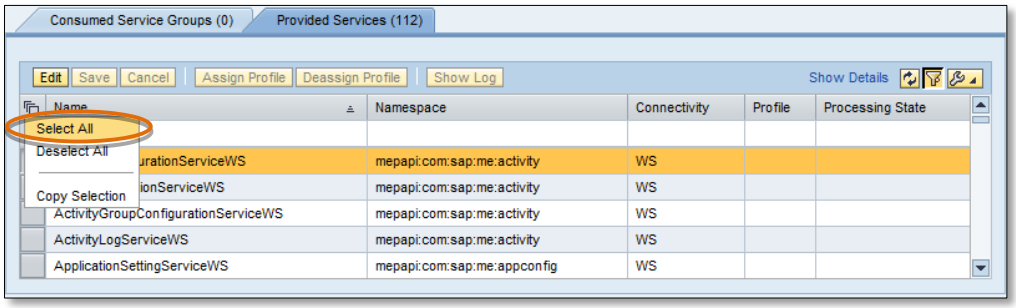
3. Choose the *Application Communication* link in the middle-left of the screen.



4. When the *Application Communication* screen appears, click the right-hand scrollbar until you see `sap.com/me~papiws` and select it by clicking the row.



5. In the lower-section of the screen, choose the *Provided Services* tab.
   A list of all services available within the `/me~papiws` package appears.
   In the following steps, we will associate these to the *Communication Profile* we created previously.
6. On the *Provided Services* tab page, click the square box in the upper-left corner of the table showing the service listing.
   A pop-up window appears.
7. Choose the *Select All* option to select all the services in the list.



8. Choose the *Edit* button on the *Provided Services* tab page.
9. Choose the *Assign Profile* button.
   A dialog appears showing the list of available *Communication Profiles* available to you.
10. Select the profile you created in the previous section, `WS_PAPI_BASIC` and choose the *OK* button.
11. Choose the *Save* button to save the associated *Communication Profile* with the PAPI web services.

    This concludes the assignment of the PAPI web services to the *Communication Profile*.

Web Service Security

When you configure accessibility to the PAPI web services, you typically do so using a *Communication Profile*. A Communication Profile provides an efficient means of setting security options that apply for all PAPI web services associated with the profile. In some cases, you may need to apply more advanced security options than what was demonstrated in the prior sections, *Creating a Communication Profile* and *Apply Communication Profile*.

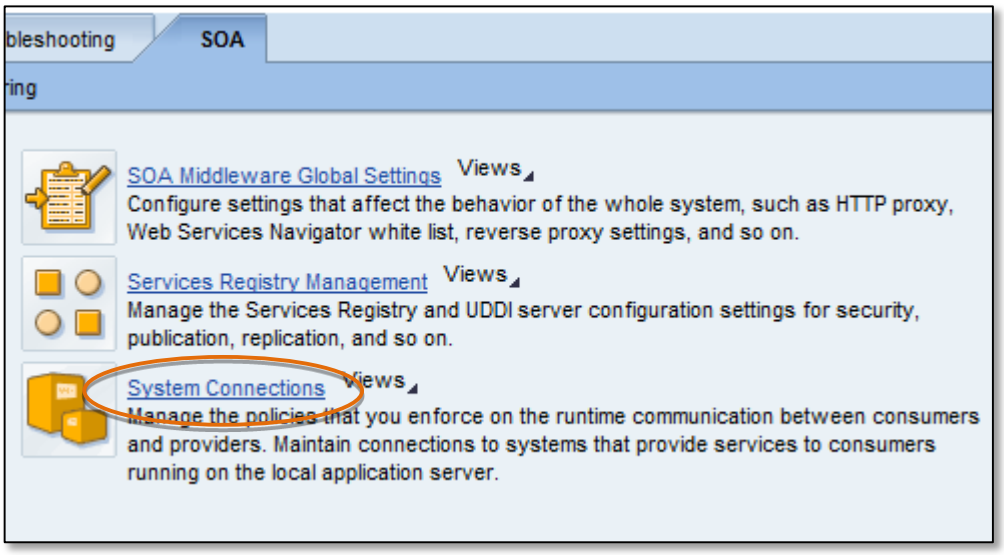
The settings that may be required largely depend on your own corporate security policy and are therefore out of scope for this document. However, if you need to make changes to an existing *Communication Profile* (or *Service Group*), the following can assist you on where to make those changes.

For more information on the *Service Group/Communication Profile* security settings, see Security Considerations for Service Groups in SAP Library for SAP NetWeaver 7.5.

To edit an existing *Communication Profile (or Service Group)* do the following:

1. In SAP NetWeaver Administrator, choose the *SOA* tab.

2. Click the *System Connections* link on the lower-right.



3. Choose the *Communication Profiles* tab.
4. Select the name of the *Communication Profile* and choose the *Edit* button.
5. Choose the *Next* button to proceed to the *Connectivity Type* configuration.
   Here you can change the *Authentication Method* and *Transport Security* options or make any other changes needed for your organization.
6. Once all changes have been made, choose the *Finish* button.
   The system activates your changes against the *Communication Profile* you selected.

See also *Configuring Web Services Security* in *SAP ME Security Guide*.

Choosing the WSDL

There are several web services exposed within the PAPI framework which developers can use to build their own applications. These services are exposed as WSDL's or (web services description language) which is an XML document that describes network services as endpoints. The WSDL identifies the service and operations (or methods) that are available for use by the consumer or application.

The service that you will reference largely depends on the function you wish to perform within SAP ME. There are two basic types:

- ***Configuration Services*** – These services are typically used for the creation or modification of data. These methods typically end with the name create, update, read or delete. These services also commonly include a finder or read method. That is, they are used to retrieve one (or more) sets of configuration data based on a supplied object for filtering the data required in the response. Finder methods are useful when you need to retrieve the reference (or handle) for an object. It may be used to locate the reference for an Operation, Material or any number of configuration-related references or objects.

  RECOMMENDATION

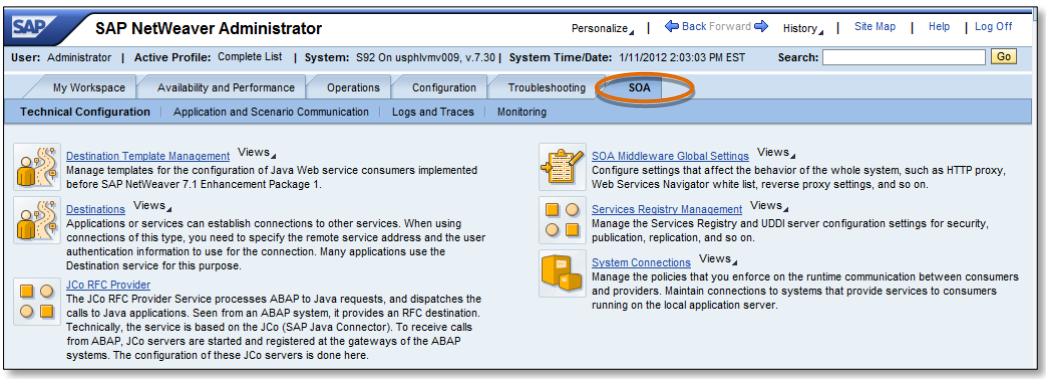
  Do not create references programmatically but instead utilize a finder to retrieve the reference value.

- ***Production Services*** – These services provide the ability to perform some action within SAP ME. That action may be to start an SFC on the shop floor, complete, send parametric data and so on. See the Javadoc for your version of SAP ME to determine the service needed for your specific application. Usage of each Web service is beyond the scope of this document.

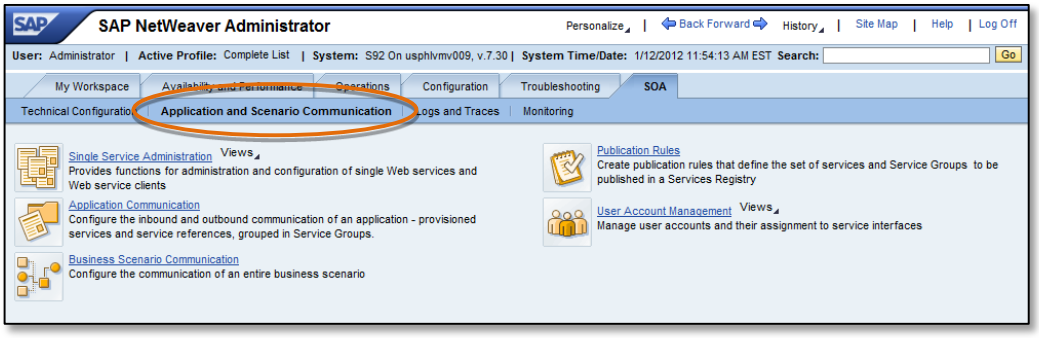
Now that you have an understanding of the types of services that exist in the PAPI web services, you may be wondering how you locate the WSDL itself. The WSDL can be viewed within your Internet Explorer (IE) browser and is an XML document.

After you have examined the Javadoc and know which service you would like to access in your application, use the following information to assist you in determining the full WSDL path to the PAPI web service:

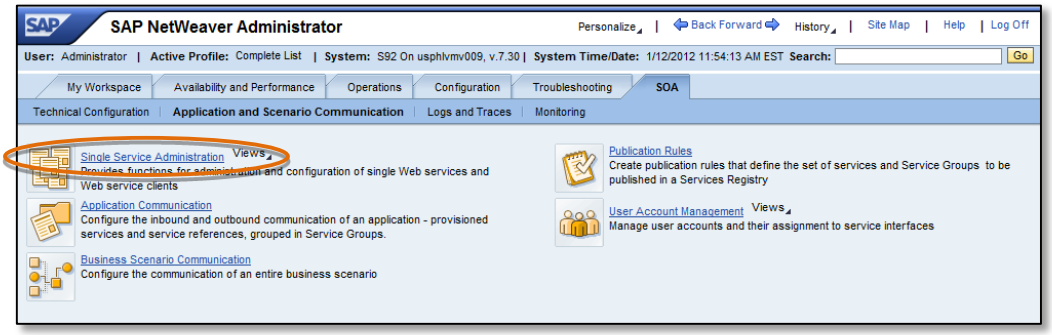
1. In SAP NetWeaver Administrator, choose the *SOA* tab.



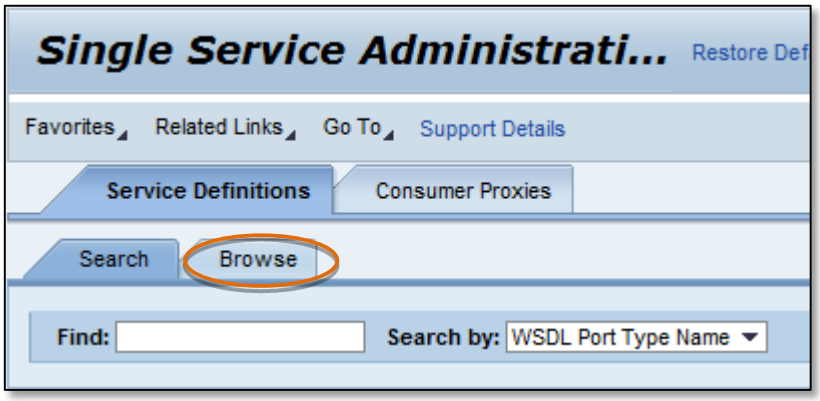
2. On the SOA tab page, click the *Application and Scenario Communication* link.



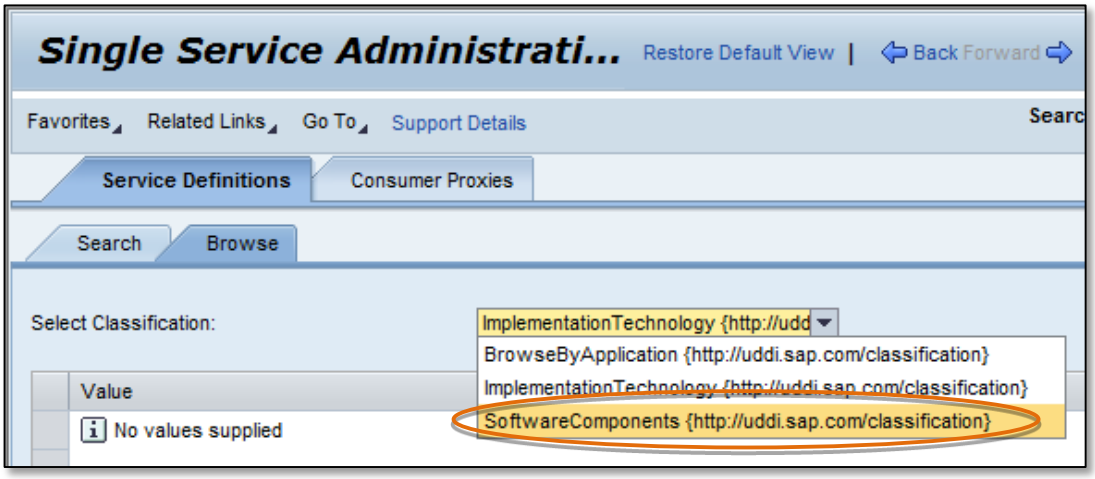
3. Choose the *Single Service Administrator* link.



4. On the *Single Service Administration* screen, on the *Service Definitions* tab page, choose the *Browse* button.

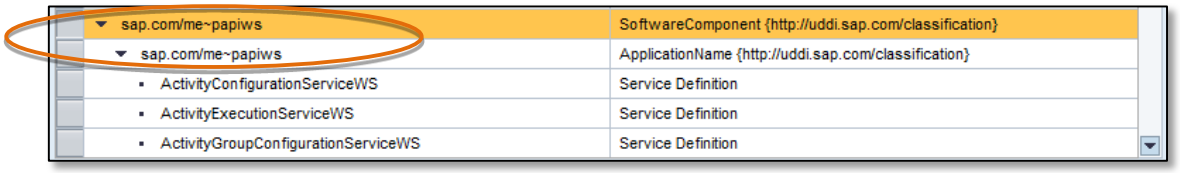


5. Click the drop-down to the right of *Select Classification* and select the option for *Software Components*.

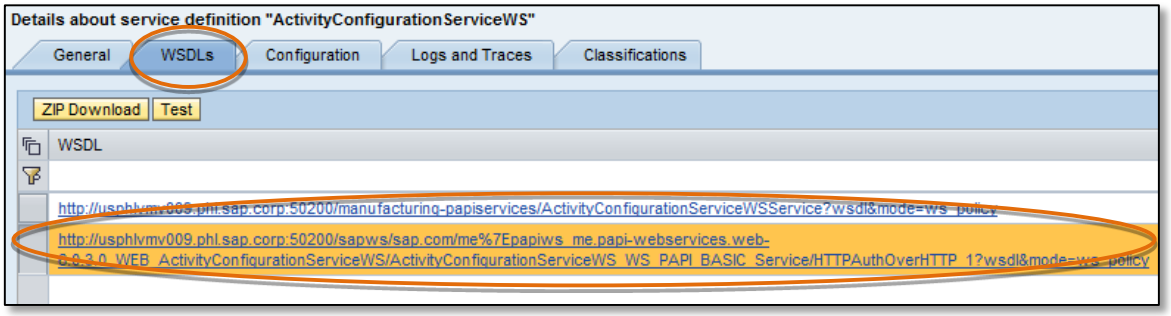


A list of components appears.
The one we are interested in is `sap.com/me~papiws`.

6. Browse the list of *Software Components* and expand (click the ▸ icon to the left of `sap.com/me~papiws`).



7. To locate the WSDL path, you may select any of the services under the `sap.com/me~papiws` branch.
   In our case, we will simply use the first one in the list, *ActivityConfigurationServiceWS*.
   Click the *ActivityConfigurationServiceWS*, when you do you will notice some additional tabs appear at the bottom of the page.
8. Choose the WSDLs tab.
   The full WSDL path(s) available in the table below appears.



9. The WSDL path that we are interested in is the one that includes our *Communication Profile* that was created in the earlier section, `WS_PAPI_BASIC`. You can click the link; this will launch your browser with the WSDL address in the address bar of your browser along with some XML in the body of page identifying binding details. The full URL in the case of this example will resemble the following:

   *http://<host>:<port>/sapws/sap.com/me~papiws_me.papi-webservices.web-<SAP ME version>_WEB_ActivityConfigurationServiceWS/ActivityConfigurationServiceWS_**WS_PAPI_BASIC**_Service/HTTPAuthOverHTTP_1?wsdl&mode=ws_policy*

See also *Configuring Web Services Security* in *SAP ME Security Guide*.

This concludes this section on choosing the WSDL.

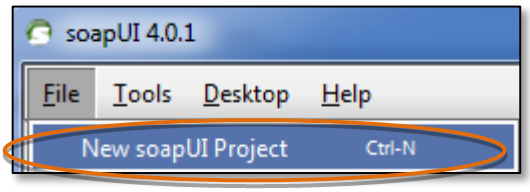
Sending a PAPI Web Service Request

As mentioned previously, there are many development environments available for building and invoking the PAPI web service requests. If you plan to implement your own custom applications, you may find using a special-purpose tool for sending web service requests to SAP ME as an ideal solution to make sure you understand the requirements for a given project. There are a number of tools available to allow you to do this, however one that we will cover in this section is a tool called soapUI.

soapUI is an open source tool for cross-platform Functional Testing of SOAP requests. It allows you to rapidly create a Web service request, populate that request with values and send it to the SAP ME PAPI web service. This makes it an ideal starting point for any new development projects that utilize the PAPI web services. The following section will walk you through the use of soapUI against one of the PAPI web services so you can see exactly what the request and response will look like coming back from the service.

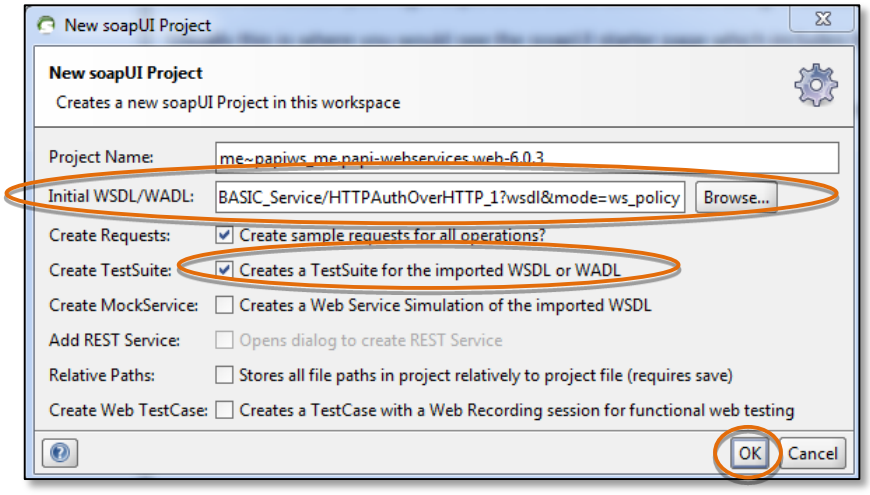
Before you begin this section, it is recommended that you review the previous section, Choosing the WSDL, and follow the steps there for determining the WSDL for the *ActivityConfigurationServiceWS*. Once you have done this, please proceed with the following:

1. Download and install soapUI.
   Visit http://www.soapui.org and download your copy of soapUI. Following the installation instructions, install soapUI on your machine.
2. Open soapUI by double-clicking the soapUI Icon from your Desktop or from the "*All Programs/SmartBear*" program group.
3. Choose *File → New soapUI Project*.

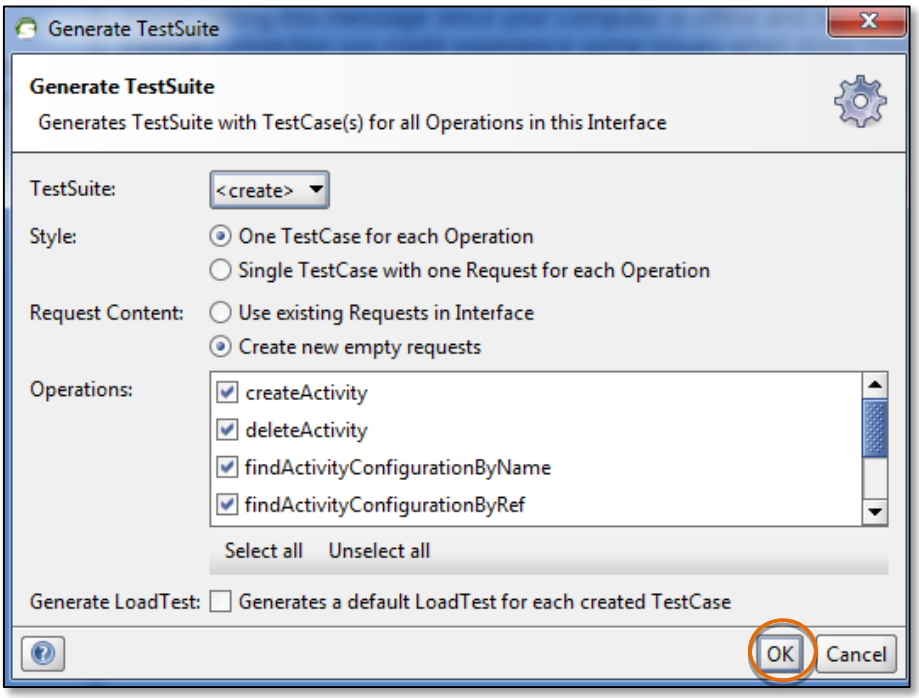


4. Enter the *Project Name* and the *Initial WSDL/WADL*.
   In our case, we want to copy/paste the WSDL for the *ActivityConfigurationServiceWS*. Your WSDL path may vary, so please use the WSDL path you found in the prior section, *Choosing the WSDL*. Once you have copied the path to the clipboard, past it into the Initial WSDL/WADL field in the dialog. You may, alternatively, change the project name to something more meaningful such as "ActivityConfigurationService-project".

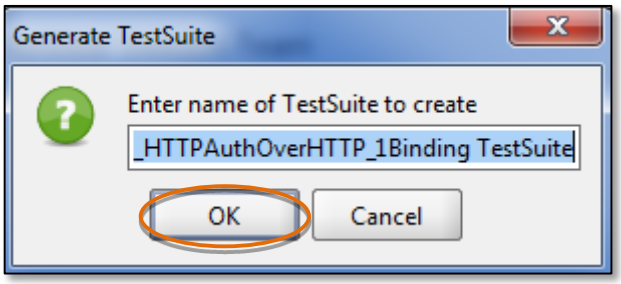
   We recommend that you select the *Creates a TestSuite for the imported WSDL or WADL* checkbox.
   Once these actions are done, click the *OK* button.



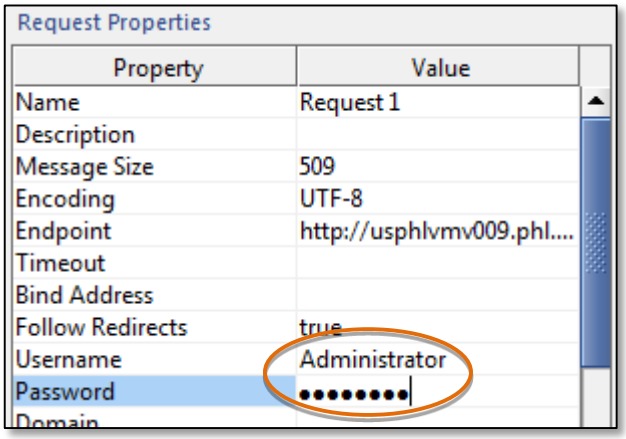
5. After a brief delay while the tool is downloading the WSDL, a dialog similar to the one below appears.
   Click the *OK* button to proceed.



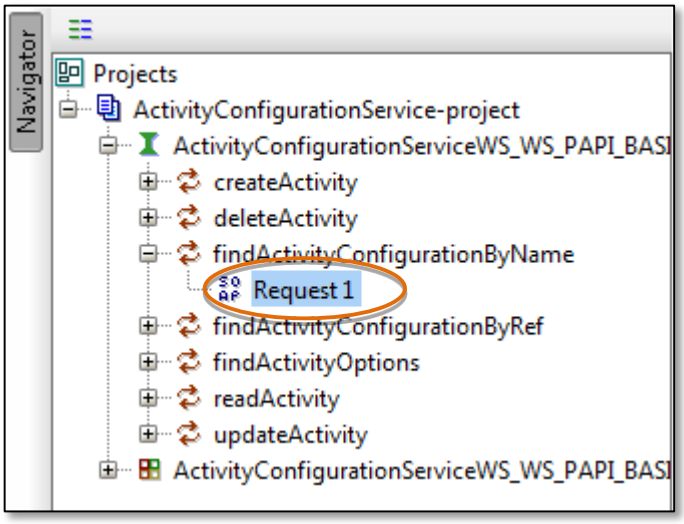
6. Once you click *OK*, you may be prompted to enter a name for the *TestSuite* to be created, you can use the default value or change it. When done, click the *OK* button.



7. In the lower-left frame, enter the web service *Username* and *Password* you wish to use. If you have not created a new Username you can use the Administrator account and password.



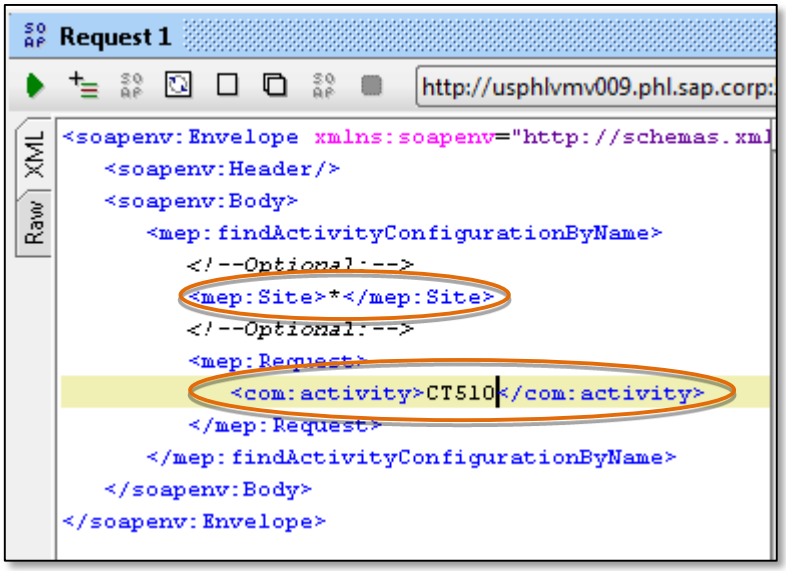
8. On the left-hand *Navigator* frame, double-click/expand the *findActivityConfigurationByName* operation (or method) and double-click the *Request1* below it.
   This will open the editor window with a pre-populated *findActivityConfigurationByName* request.
   Notice there are two values of "`?`" automatically entered for you. We will change these values in the next step.



9. We are almost ready to test the request. However, as mentioned in the previous step we have two question marks (`?`) that we need to populate with actual values. Whenever you see these values in soapUI, you need to either remove them (if they are marked as optional) or replace them with actual values. For the purpose of this procedure, you can use your own site name (if you know it) or use the Global site of `*`. That is, where you see the `?` to the right of `<mep:Site>` replace the `?` with a `*`. Do the same with the `?` to the right of `<com:activity>`, changing it to `CT510`.

   NOTE

   The value `CT510` is the activity ID for *As-Built Configuration* within SAP ME. We could have used any activity available in *Activity Maintenance* for this example.



10. We can now send the request to SAP ME.
    Click the green *Submit Request* button ( ) to send the request to the PAPI web service/SAP ME.
    The response of the web service call appears in the right-hand panel.
    It looks similar to the following response below.
    Notice the response includes a ref (Reference to the object), description, class/program and visibility setting.

11. You can try experimenting with other services as well. This was simply an introduction to the PAPI web services.
    Keep in mind, that you should read the Javadoc for the related web service you plan on using so you know exactly what data is expected and what the outcome will be sending the request. Some requests will be more complicated than other, we recommend that you begin with the finders/readers and then progress from there.

Object Resolver

In many cases you will need to acquire the reference object (also known as a handle) to be able to use it within your request messages. In most cases, a finder provides you with this information. However, to simplify your code and streamline the results of your request the *Object Resolver* may be a more ideal means of retrieving the ref.

You can use soapUI or your preferred development environment to create a request for the ObjectResolver. In the Application Development portions of this guide (next section) we demonstrate how to use the `findOperationConfigurationWS` to retrieve the configuration for an operation. If you only need the reference to the operation object, you can also use the Object Resolver to retrieve that one object.

For this example, let us use soapUI to demonstrate how to retrieve the object reference for an SAP ME operation object. In the prior section we demonstrated how to use soapUI for creating your first request. We will expand on that here and use the Object Resolver to return only the object reference for the operation:

1. Create a new soapUI project and reference the WSDL (your path to the WSDL should look similar to http://[host]:[port]/sapws/sap.com/me~papiws_me.papi-webservices.web-<SAP ME version>_WEB_ObjectResolverServiceWS/ObjectResolverServiceWS_WS_PAPI_BASIC_Service/HTTPAuthOverHTTP_1?wsdl.
2. Assign the web service a username and passwords as we did in the prior section (click *Request1* and enter the values in the properties window).
3. Double-click the *Request1* sample request under the `ObjectResolverServiceWS_.../resolveObjectReferenceByKeys` in the soapUI *Navigator*.
4. Assign a value for the *Site* parameter.
5. For the `attribute`, enter a value of `operation`.
6. For the `value`, enter the name of the operation you wish to look up, for example `OP1`.
7. For the `revisionEnum` property, enter `CURRENT` to use the current revision. The currently available `revisionEnum`'s are: `CURRENT`, `ANY` and `DEFINED`.
8. For the `tableEnum`, use the value `OPERATION` for this field. The table below may also be a helpful reference of the tables that are available to you:

ALARM_LOG	DATA_FIELD	MESSAGE
APPLICATION_SETTING	ROUTER	MESSAGE_LOG
ATTACHMENT	SHOP_ORDER	NEXT_NUMBER
ATTENDANCE_LOG	ROUTER_STEP	OPERATION
BACKGROUND_PROCESS	SFC	PROCESS_LOT
BOM	WORK_CENTER	PRODUCTION_LOG
BUYOFF	RESOURCE_TYPE	REASON_CODE
BUYOFF_LOG	CUSTOMER_ORDER	SAMPLE_PLAN
CERTIFICATION	DATA_TYPE	WORK_INSTRUCTION
CONTAINER	DC_GROUP	ITEM
CONTAINER_DATA	INVENTORY	RESRCE
COST_CENTER	INVENTORY_LOG	
CUSTOMER	ITEM_GROUP	

NOTE

This list may change and you should therefore consult the Javadoc for the latest information.
To find this information in the Javadoc, do the following:

1. Choose the `com.sap.me.common` package.
2. In the lower-left navigation pane, choose `ObjectAliasEnum`.

Your completed request should look similar to the following:

EXAMPLE

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:mep="mepapi:com:sap:me:common">
   <soapenv:Header/>
   <soapenv:Body>
      <mep:resolveObjectReferenceByKeys>
         <mep:Site>test</mep:Site>
         <mep:Request>
            <keyFieldList>
               <attribute>operation</attribute>
               <value>OP1</value>
            </keyFieldList>
            <revisionEnum>CURRENT</revisionEnum>
            <tableEnum>OPERATION</tableEnum>
         </mep:Request>
      </mep:resolveObjectReferenceByKeys>
   </soapenv:Body>
</soapenv:Envelope>
```

9. Send the request to SAP ME and return the object reference. To do this, click the green *Submit Request* button ( ).
   You will receive a response similar to the following:

   EXAMPLE

```
<SOAP-ENV:Envelope xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Body>
      <ns2:resolveObjectReferenceByKeysResponse
xmlns:ns2="mepapi:com:sap:me:common">
         <ns2:Response>
            <ref>OperationBO:TEST,OP1,A</ref>
         </ns2:Response>
      </ns2:resolveObjectReferenceByKeysResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

   Notice the section in bold in the response. This is the object ref for the operation `OP1`. You can experiment with other objects to see exactly how they work. For the most current information, see the *SAP ME Javadoc*.

DateTime Usage

Some PAPI web service requests may need to include a `DataTime` formatted according to the W3C data type. The format for `DateTime` fields must be done as follows:

The `DateTime` core component type uses the W3C built-in data type `xsd:dateTime`. This is structured in accordance with the extended representation of ISO 8601 (see NOTE-datetime). However, unlike in `xsd:date`, it is not possible to represent negative years or years with more than 4 numeric values in "Date".

For Java, the following code format should be used:

```
java.text.SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ssZ").format(new Date())
```

The extended representation is as follows:

```
CCYY-MM-DDThh:mm:ss(.sss)(Z)
```

or

```
CCYY-MM-DDThh:mm:ss(.sss)(+/-)hh:mm
```

Example:

```
2002-04-19T15:30:00Z
```

or

```
2002-04-19T10:30:00+05:00
```

The extended representation uses the following literals:

Code	Description	Value Range
`CC`	for century	00 - 99
`YY`	for year	00 - 99
`MM`	for month	01 -12
`DD`	for day	01 - 28 for month 02; 01 - 29 for month 02 when the year is a leap year; 01 - 30 for months 04, 06, 09, and 11; 01 - 31 for months 01, 03, 05, 07, 08, 10, and 12
`-`	A hyphen between the year, month, and day is mandatory.	
`T`	A separator between the date and time is mandatory.	
`hh`	for hours	00 - 23
`mm`	for minutes	00 – 59
`ss`	for seconds	00 – 59
`sss`	One or more characters after the decimal point represent fractions of a second. The representation is limited to a maximum of three decimal places, that is, the time can be expressed to a thousandth of a second.	
`:`	A colon between the hours, minutes, and seconds is mandatory.	
`Z`	must be specified when the represented time is also the UTC time.	

Code	Description	Value Range
`+hh:mm`	must be specified when the represented time is a local time that is ahead of UTC time.	
`-hh:mm`	must be specified when the represented time is a local time that is behind UTC time.	

Note: For more detailed specification, see:
http://wiki.sdn.sap.com/wiki/display/DataIntegration/SAP+CDT+-+DateTime

Application Development

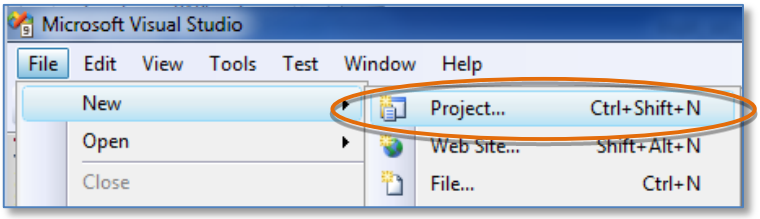
The reason for exposing the SAP ME API's within the PAPI web service framework is to allow organizations to harness the feature set of SAP ME in a technology agnostic manner. For instance, you can utilize environments such as Java or the .NET framework to create applications that meet your specific needs. In this section, we will cover how to develop an application to retrieve configuration information from SAP ME via the PAPI web services. The first example demonstrates how to accomplish this goal within the .NET development environment using Microsoft Visual Studio 2008. The second example functions almost identically and is written in Java.

Creating a Project

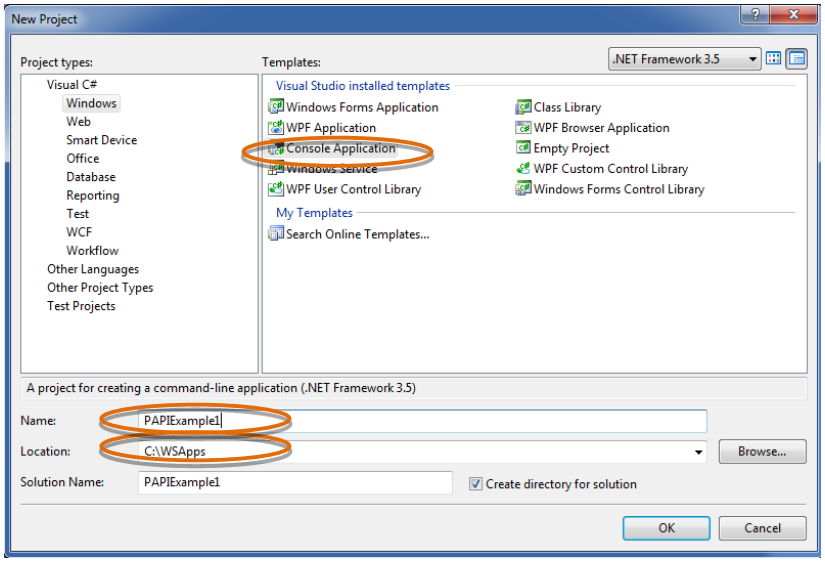
The first step is creating your .NET project in Microsoft Visual Studio 2008. Proceed as follows.

1. Start Microsoft Visual Studio.

2. Select the menu option *File* → *New* → *Project* as depicted in the screenshot below.



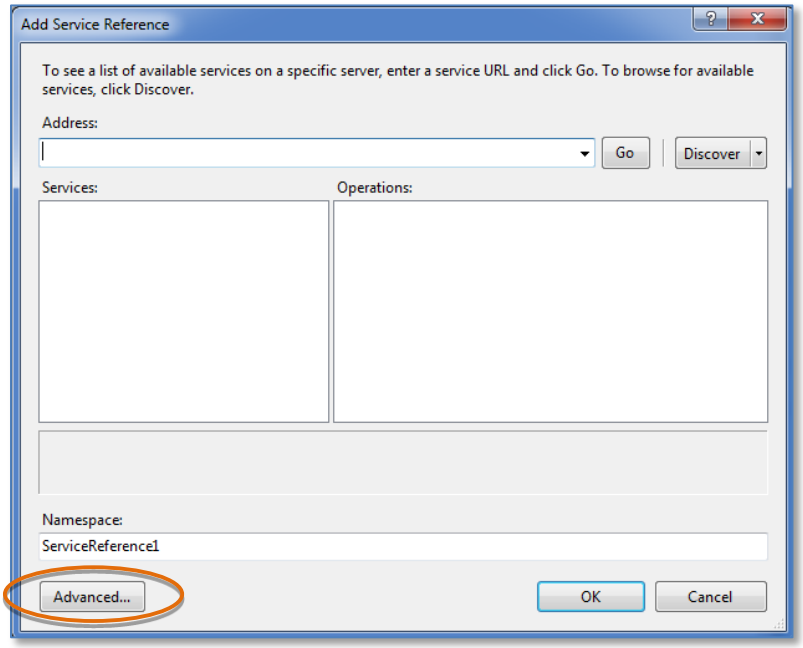
3. Select the *Console Application* template from the list of available templates. Select the folder where you would like your project created (i.e. WSApps) and enter a name for your project, such as *PAPIExample1*. Once completed, click the *OK* button.



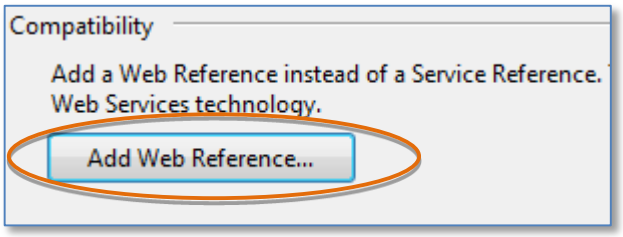
Creating a Service Reference

Now that we have the start to our application, we need to create a service reference to the services we wish to utilize within our application. For this example, we will utilize the *OperationConfigurationServiceWS* to query it for details of an operation.
Before proceeding with this section, it's a good idea to have SAP ME configured and know the site, operation and web service user and password you wish to use.

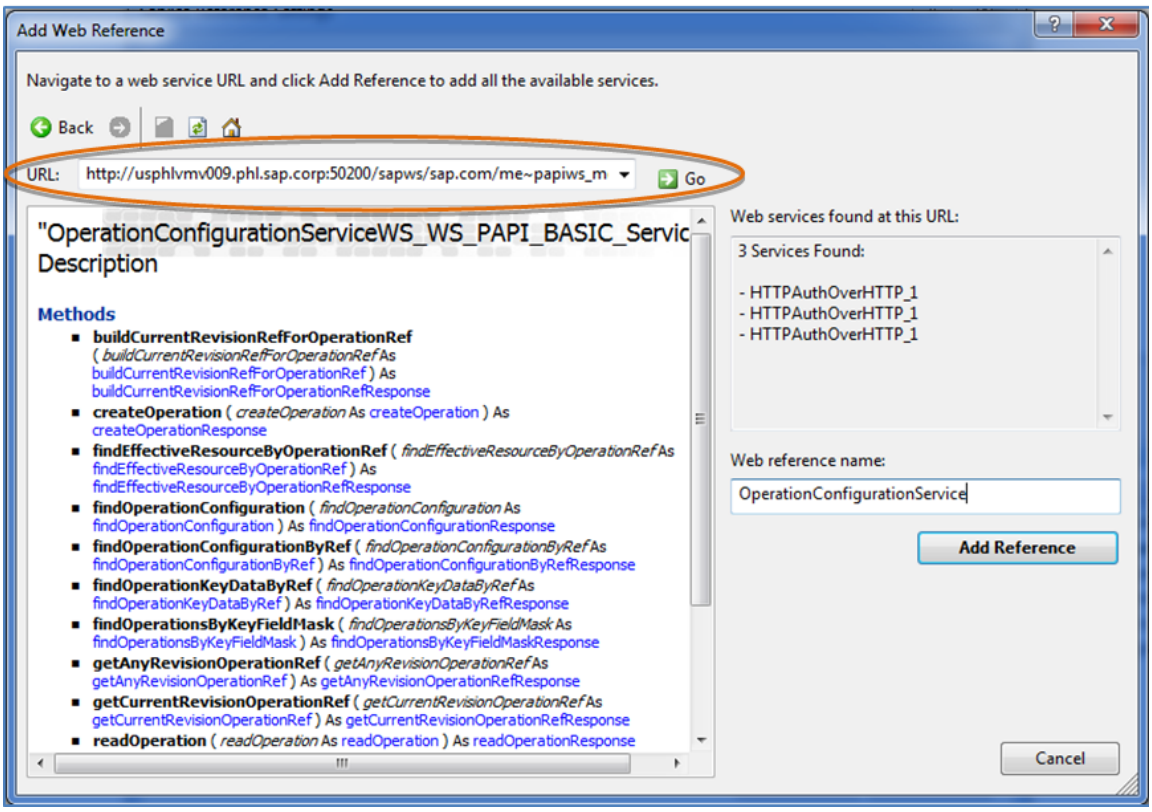
4. From the *Project* menu option, select *Add Service Reference*. Then choose the *Advanced* button.



5. Choose the *Add Web Reference* button.



6. Provide the full path to the WSDL (see the prior section entitled *Choosing the WSDL*" if you are unsure of the WSDL for the *OperationConfigurationService).*
Copy and paste the path to the WSDL in the *URL* field and choose the *Go* button to discover the web service and enumerate the methods that are implemented in that particular web service.



7. For the *Web reference name*, replace the value with a more meaningful name such as `OperationConfigurationService`, and then choose the *Add Reference* button. The reference has now been added and you now have access to the objects contained within the service from your Visual Studio project.

Coding the Example

Now we can write some code.

8. Double click the `Program.cs` class from your project. There is a method that resides here called *Main* and as you would expect is the Main entry method of the program.
9. Our first step in preparing the code is to add the `using` statements that will allow access to the *OperationConfigurationService*. At the top of your `Program.cs` class you will see a few boilerplate `using` statements.

   Add the following to the bottom of that list:

   SYNTAX

   ```
   using System.Net;
   using PAPIExample1.OperationConfigurationService;
   using System.Web.Services.Protocols;
   ```

10. To simplify the logging of the messages to the console window, we will now apply a small private log method to use the `Console.WriteLine()` to log our messages to the Console.

    Add the following code below the `class Program {` statement:

    SYNTAX

    ```
    // Log a Message to the Console
    private static void Log(string msg)
    {
          Console.WriteLine(msg + Environment.NewLine);
    }
    ```

11. Define the objects used to store the information to query the PAPI web service. Such information includes strings to store the site, operation and object for the Network Credentials.
    Apply the following code to the beginning of the Main method in the `Program.cs` class:

    SYNTAX

    ```
    // Change the values here to match your environment.
    string site = "TEST";            // The SAP ME Site
    string operation = "OP1";        // The SAP ME Operation
    string user = "SITE_ADMIN";      // The Web service user account
    string password = "Your Password Here";     // The Web service password
    NetworkCredential verifiedLogon = null;     // Contains the Web service
    logon credentials
    ```

    NOTE

    All code from this point forward must be added directly below the code in prior sections. That is, you can simply copy/paste this code as it is explained directly below the previous code.

12. Assign a value to the *NetworkCredentials* object. This object contains the `verifiedLogon = new NetworkCredential(user, password, null);` authentication information needed for the web service request.

    After the prior code, add the following:

SYNTAX

```
OperationConfigurationService.OperationConfigurationServiceWS_WS_PAPI_BASIC_Service operationConfigurationProxy = new OperationConfigurationServiceWS_WS_PAPI_BASIC_Service();
operationConfigurationProxy.Credentials = verifiedLogon;
operationConfigurationProxy.PreAuthenticate = true;
```

13. Create an instance of the web reference proxy that will be used to send the web service request. We will call this object the *operationConfigurationProxy*. Then set two properties, one containing the *verifiedLogon* (NetworkCredentials), the other property to set the request to pre-authenticate to true.
Pre-authentication reduces the amount of network traffic used to authenticate the request.

SYNTAX

```
findOperationConfiguration findOperationConfigurationRequest = new findOperationConfiguration();
```

14. Create the request message itself.
For the *findOperationConfiguration* request, we will create that request object u the following code:

SYNTAX

```
findOperationConfigurationRequest.Site = site;
findOperationConfigurationRequest.Request = new OperationSearchRequest();
findOperationConfigurationRequest.Request.operation = operation;
```

15. Populate the *findOperationConfiguration* request with the properties needed to perform our query against the Operation. To do this, we will first assign the site (String), create a new Request(*OperationSearchRequest*) object and provide it with the operation that we wish to search (String).

16. Add the code to send the request and return the *findOperationConfigurationResponse* object.
We will encapsulate the request within a try/catch block. This will allow us to identify any exceptions returned and post the reason for the exception to the console(We will cover the exceptions in the next section). We will first create the *findOperationConfigurationResponse* object and invoke the *findOperationConfiguration* method against the *operationConfigurationProxy*. We will then use an `if` statement to check if the response was not null and that a response was returned. If everything is successful, we simply log a message with the details of the response.

SYNTAX

```
try
{
        findOperationConfigurationResponse findOperationConfigurationResponse =
        operationConfigurationProxy.findOperationConfiguration(findOperationConfigurationRequest);

        Log("Find successful.  Details:");
        if (findOperationConfigurationResponse != null && findOperationConfigurationResponse.Response.Length > 0)
        {
```

```
                Log(string.Format("Ref: {1}{0}Operation: {2}{0}Revision:
        {3}{0}Current Revision: {4}{0}",
                 Environment.NewLine,
                findOperationConfigurationResponse.Response[0].@ref,findOpe
        rationConfigurationResponse.Response[0].operation,

findOperationConfigurationResponse.Response[0].revision,
      findOperationConfigurationResponse.Response[0].currentRevision)
                          );
            }
}
```

Exception Handling

Exception handling is the act of catching possible errors that may be encountered within an application. In the case of the PAPI web services, and all web services for that matter, there are specific conditions that you will want to "catch". The *SoapException* and *WebException* classes should be of particular interest within your application. The following steps demonstrate how you can catch those exceptions within the example:

17. The `SoapException` is thrown in the event a web service method is called and results in an exception. Therefore, add the following code below the "try" block to catch SOAP exceptions and log the reason for the exception to the console.

    SYNTAX

```
catch (SoapException ex)
{
        Log(ex.Message.ToString());
}
```

18. A second exception we want to catch is the *WebException*. This exception is thrown when an error occurs while accessing the network or resulting in a protocol exception. Add the following exception handling code below the `SoapException` to handle the WebException condition.

    SYNTAX

```
catch (WebException ex)
{
        HttpWebResponse response = (HttpWebResponse)ex.Response;

        switch (response.StatusCode)
        {
                case HttpStatusCode.Unauthorized:
                        Log("Username or password is not correct.\nPlease
                verify and try again.");
                        break;
                default:
                        Log(ex.Message);
                break;
        }
}
```

19. The last bit of code for this example is simply to request the user to press the ENTER key to terminate the application.
    To do this, add the following code to await the ENTER key to be pressed.

SYNTAX

```
Console.ReadLine();
```

That completes this code example. See Appendix A: Source Code Listing for a complete source code listing for this example if you have any problems.
An archive is also included with this guide that contains the full Visual Studio 2008 solution for you to use as a basis for your own applications.

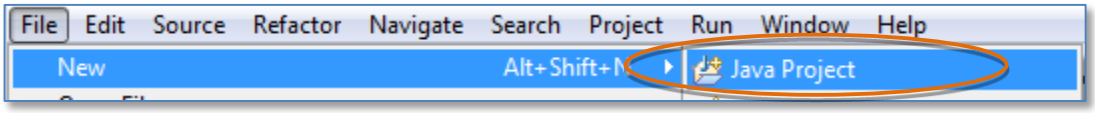
Java Example

This section demonstrates how to create a simple Java application which invokes the *findOperationConfiguration* method and returns the operation details. Before coding this example, be sure you have an existing SAP ME site and operation available for use. Additionally, you will need to know the Web service user and password. The IDE we will use for this example is SAP NetWeaver Developer's Studio (NWDS). If you do not already have NWDS installed on your machine, you will need to do so to follow along with this example.

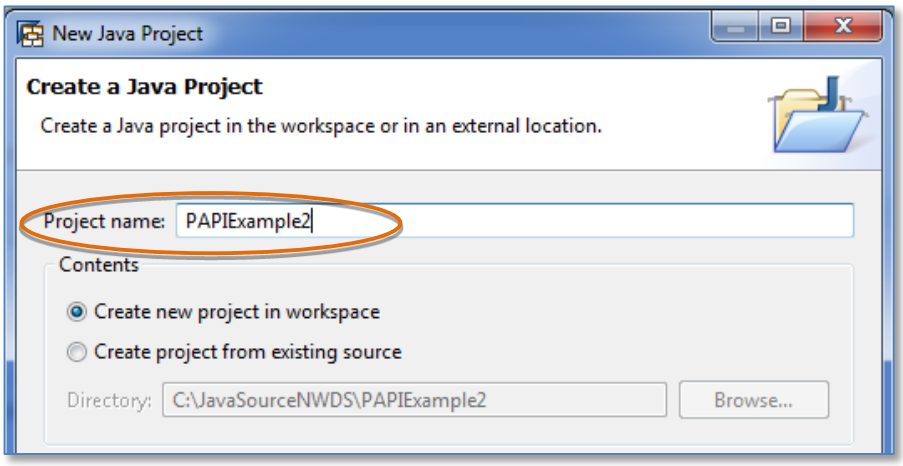
Creating a Project

The first step is to launch NWDS and create your project. The steps below will walk you through how to accomplish this:

1. Start SAP NetWeaver Developer's Studio by double-clicking the *SAP NWDS icon* (if you have created one) or using explorer browse to the location where NWDS is installed and double-click the executable, *SapNetweaverDeveloperStudio.exe*.
2. From the menu bar, choose *File → New → Java Project* as shown in the screenshot below.



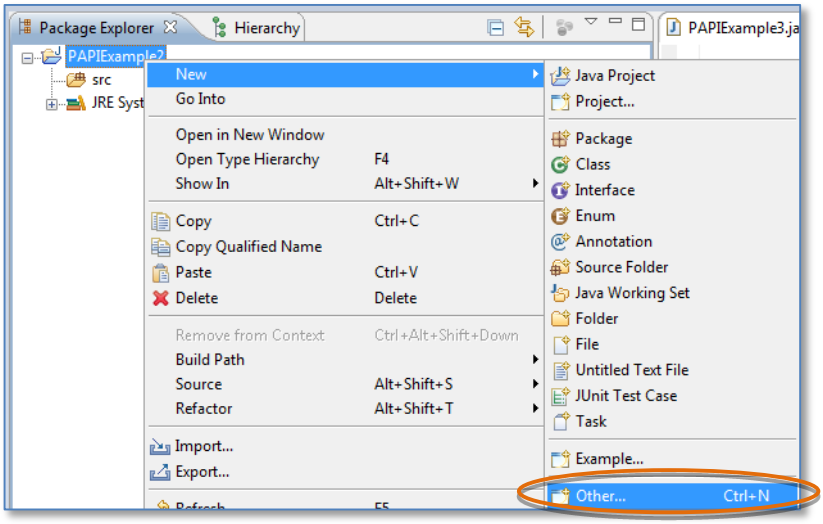
3. Enter a name for the application such as *PAPIExample2*. Once completed, choose the *Finish* button.



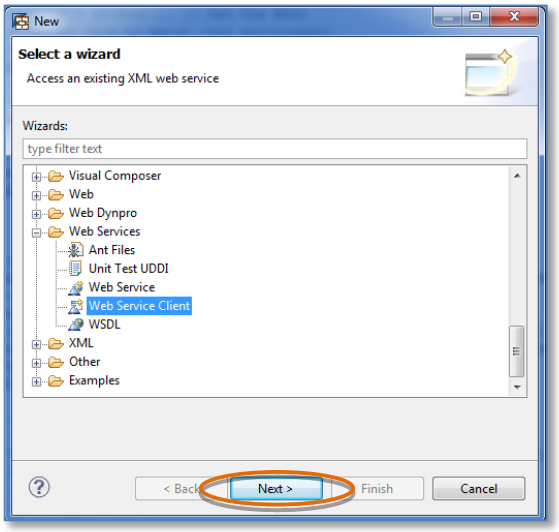
Creating a Reference to the WSDL

Now that we have an empty project, we need to create a reference to the service(s) we wish to utilize within our application. For this example, we will utilize the *OperationConfigurationServiceWS* to query it for details of an operation.

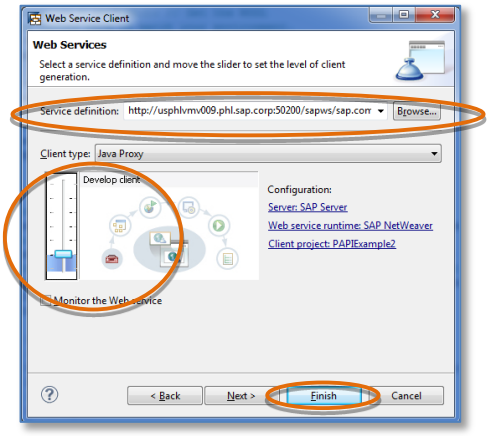
4. Right-click the project *PAPIExample2* and choose the menu option *New → Other…*



5. Add a Web *Service Client*.
   Browse the tree and expand Web *Services* and select Web *Service Client*. Once done, choose the *Next* button.



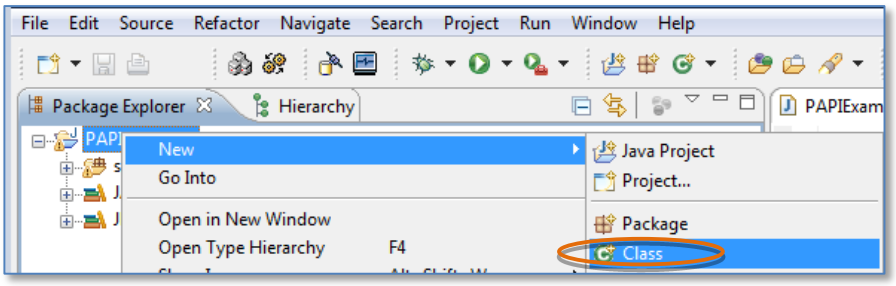
6. On the Web *Service Client* dialog, copy/paste the path to the WSDL in the *Service definition* field. Then click and hold the slider on the left-hand side and drag it to *Develop Client*. Once completed, choose the *Finish* button. After that, a brief delay will occur while the WSDL is downloaded and parsed.



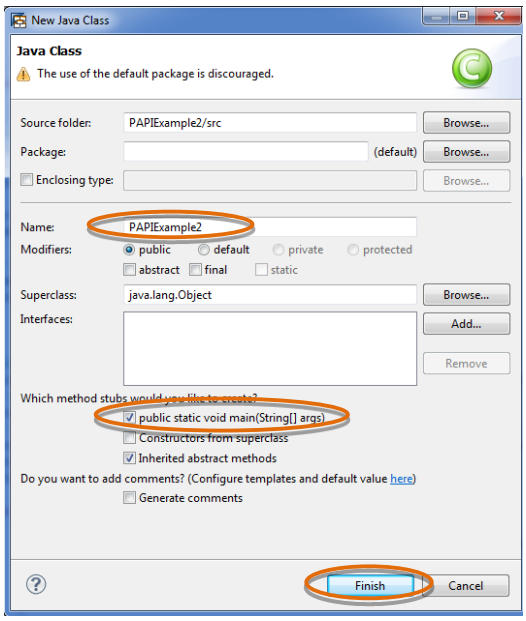
Coding the Example

We are now at the point where we will begin writing some code.

7. Add a Java Class to our project. Right-click the *PAPIExample2* folder from the *Package Explorer* and choose *New → Class*.



8. Enter a name for the class such as *PAPIExample2*.
   For the option asking "*Which method stubs would you like to create?*" select "*public static void main(String[]args)*". Once done, click the *Finish* button.



9. Your PAPIExample2 class will then resemble the following code:

SYNTAX

```
public class PAPIExample2 {

   /**
    * @param args
    */
   public static void main(String[] args) {
      // TODO Auto-generated method stub

   }
}
```

10. Now that we have our project setup and ready, it is time to add our own code. At the top of the `PAPIExample2` class, we will add import statements for the class we need to access.

To do that, add the following lines of code to the top of the `PAPIExample2` class:

SYNTAX

```
import java.net.MalformedURLException;
import java.util.Map;
import javax.xml.ws.BindingProvider;
// Service Imports
import mepapi.com.sap.me.productdefinition.FindOperationConfigurationFault;
import mepapi.com.sap.me.productdefinition.OperationConfigurationServiceWS;
import mepapi.com.sap.me.productdefinition.OperationConfigurationServiceWSService;
// Operation Imports
import com.sap.me.productdefinition.OperationExtendedConfiguration;
import com.sap.me.productdefinition.OperationSearchRequest;
import com.sap.me.productdefinition.OperationSearchResult;
```

11. To simplify the logging of the messages, apply a small private log method to that will be used to write out the results back to the IDE or Console.
    Add the following code the import statements and before the `main()`.

    SYNTAX

    ```
    private static void log(String message) {
       System.out.println(message);
    }
    ```

    We have all the boiler-plate code completed, we will focus on the key feature of our application
12. Declare the strings used to hold the values for the site, operation, Web service username and password (change the text for the user/password to match your environment).

    SYNTAX

    ```
    String site = "TEST";
    String operation = "OP1";
    String user = "SITE_ADMIN";
    String password = "[Your password here]";
    ```

13. Create the *OperationSearchRequest* object that will contain our search request and set the value to the operation to search for.
    After the above declarations, add the following code:

    SYNTAX

    ```
    OperationSearchRequest request = new OperationSearchRequest();
    request.setOperation(operation);
    ```

14. Create an instance of the service and acquire a reference to the proxy. To do this, add the following code directly after the prior code:

SYNTAX

```
OperationConfigurationServiceWSService service = new
OperationConfigurationServiceWSService();
OperationConfigurationServiceWS proxy =
service.getOperationConfigurationServiceWSPort();
```

15. Add the code that is needed to handle the *Basic Authentication* requirements. That is, assign the user and password to the *requestContext*. To do this, apply the code after the previous code:

SYNTAX

```
Map<String,Object> requestContext =
((BindingProvider)proxy).getRequestContext();
requestContext.put(BindingProvider.USERNAME_PROPERTY, user);
requestContext.put(BindingProvider.PASSWORD_PROPERTY, password);
```

16. Before we can add the code to send the request, we need to encapsulate the code within a try/catch block. This will allow us to catch and log any exceptions that are returned. After the above code, add the following try/catch block:

SYNTAX

```
try {
        // Send the findOperationConfiguration request


// Catch any Exceptions
} catch (FindOperationConfigurationFault e) {

            // Log Error
      log(e.getMessage());
}
```

17. The block of code between the `try { }` is the location where the next several lines of code will be placed. Continue to place the code here until otherwise instructed. Our next step is to create the *OperationSearchResult* object which will contain the response of the *findOperationConfiguration* request.
Add the following line immediately after the `try {` code in the previous step:

SYNTAX

```
response = proxy.findOperationConfiguration(site, request);
log("Find successful.  Details:\n");
```

18. After sending the *findOperationConfiguration* request, we now will display the results of the request back to the user.
Create an `if` statement and check if the response is not null. If it is not, iterate through the `response.getOperationList()` results and display them using our `Log()`. In the else, simply log a message indicating the response was null. The following is what the code should look like:

SYNTAX

```
if (response != null)
{
   for(OperationExtendedConfiguration c: response.getOperationList())
   {
      log("Ref: " +c.getRef());
      log("Operation: " + c.getOperation());
      log("Revision: " +c.getRevision());
      log("Current Revision: " + c.isCurrentRevision().toString());
   }
} else
{
      log("Response was null.");
}
```

Exception Handling

There is one very important piece of code missing. The missing code is that which will check for a malformed URL Exception and handle it properly. A malformed URL is one that does not follow the standard URL structure (i.e. `http://`).
Proceed with the remaining steps to complete the *PAPIExample2 project*.

19. Add a Try block above the prior code starting with "`OperationConfigurationServiceWSWSPAPIBASICService service =…`". Then apply the catch statement as defined below to log the exception to the IDE/Console.

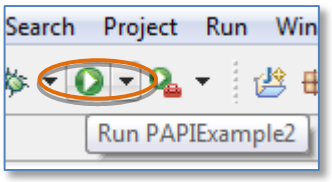
 SYNTAX

```
try {



} catch (MalformedURLException murle)
{
   log("Exception: " + murle.getMessage());
}
```

20. Choose the *Run* button from the NWDS toolbar to run your program.



After running the application, you should receive a response that looks similar to the following:

```
Find successful.  Details:
Ref: OperationBO:TEST,OP1,A
Operation: OP1
Revision: A
Current Revision: True
```

That concludes this example.

See Applendix A: Source Code Listing for a full source listing, if needed.

Troubleshooting

Sometimes things go wrong, most commonly during testing of your web services. Often, the problem is easily resolved by examining the error response that was returned and making a change. It may be as simple as an incorrect web service user account or an invalid password. Although, sometimes the issues you encounter may be more complex. Below are some helpful problems and potential solutions to get you on the right track:

Problem	Source	Reason	Solution
Received a response of : “<some message> (Message <some number>)”	Business Rule Violation	Data you sent violated the business rules.	Correct the data being sent in the request and re-try.
Timeout Exception	Communication Error	The request did not get processed in a timely fashion and timed out.	Increase the timeout parameter on the client or examine the server logs for clues to the delayed response.
Response “Error 4xx”	Client Application	Varies	A client application error has occurred.
Response “Error 5xx”	Server error	Varies	An error has occurred at the server, check the associated system logs.

If the above information was not able to help, you may find a packet sniffing application helpful for seeing exactly what messages are being sent across the wire. One of many such applications is WireShark. WireShark does not require you to setup a proxy (i.e. change ports) and is therefore very useful for debugging those complex communication errors.

Appendix A: Source Code Listing

This section contains the full source listing for both the .NET and Java example applications.

.NET Source Code Listing for PAPIExample1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Net;
using PAPIExample1.OperationConfigurationService;
using System.Web.Services.Protocols;


namespace PAPIExample1
{
    class Program
    {

        // Log a Message to the Console
        private static void Log(string msg)
        {
            Console.WriteLine(msg + Environment.NewLine);
        }

        static void Main(string[] args)
        {
            // Change the values here to match your environment.
            string site = "TEST";           // The SAP ME Site (must currently exist in SAP ME)
            string operation = "OP1";       // The SAP ME Operation you wish to query (must currently exist in SAP ME)
            string user = "SITE_ADMIN";     // The Web service user account
            string password = "your password here";    // The Web service password
            NetworkCredential verifiedLogon = null;  // Contains the Web service logon credentials

            // verify the password by requesting a list of operations defined in this SITE.
            // Create the NetworkCredential object that would be used through the lifetime of the application.
            verifiedLogon = new NetworkCredential(user, password, null);

            // Create the Proxy Object for the OperationConfigurationService (see details of how to do this in the PAPI web service guide)
            // The name of the service indicated below may not exist and may need to be regenerated (and modified) in this solution.
            OperationConfigurationService.OperationConfigurationServiceWS_WS_PAPI_BASIC_Service operationConfigurationProxy = new
            OperationConfigurationServiceWS_WS_PAPI_BASIC_Service();
            // Assign Credentials
            operationConfigurationProxy.Credentials = verifiedLogon;
            // Set to pre-authenticate, this reduced the amount of network traffic to send the request.
            operationConfigurationProxy.PreAuthenticate = true;

            // Create the findOperationConfiguration request object
            findOperationConfiguration findOperationConfigurationRequest = new findOperationConfiguration();

            // Assign the values to the request
            findOperationConfigurationRequest.Site = site;
            findOperationConfigurationRequest.Request = new OperationSearchRequest();
            findOperationConfigurationRequest.Request.operation = operation;

            // Send the request
            try
            {
                // Send the request and return the results into the findOperationConfigurationResponse object
                findOperationConfigurationResponse findOperationConfigurationResponse =
operationConfigurationProxy.findOperationConfiguration(findOperationConfigurationRequest);

                // Log the details from the response to the console (uses a simple helper function called Log to minimize redundant code)
                Log("Find successful. Details:");
                if (findOperationConfigurationResponse != null && findOperationConfigurationResponse.Response.Length > 0)
                {
                    // Log the details of the response
                        Log(string.Format("Ref: {1}{0}Operation: {2}{0}Revision: {3}{0}Current Revision: {4}{0}", Environment.NewLine,
                        findOperationConfigurationResponse.Response[0].@ref,findOperationConfigurationResponse.Response[0].operation,
                            findOperationConfigurationResponse.Response[0].revision,
findOperationConfigurationResponse.Response[0].currentRevision)
                        );
                }
            }
            catch (SoapException ex)
            {
                // A SOAP Exception was returned, log the details.
                Log(ex.Message.ToString());
```

```
            }
            catch (WebException ex)
            {
                // The WebException.Response is type of WebResponse, which is the base of HttpWebResponse.
                // For this reason, we have to cast it to child type.
                HttpWebResponse response = (HttpWebResponse)ex.Response;

                switch (response.StatusCode)
                {
                    case HttpStatusCode.Unauthorized:
                        //Authentication failure.
                        Log("Username or password is not correct.\nPlease verify and try again.");
                        break;
                    default:
                        //some other unanticipated error occured.
                        Log(ex.Message);
                        break;
                }
            }

            // Don't exit the application until the user presses [ENTER].
            Console.ReadLine();
        }

    }
}
```

Java Source Listing Code for PAPIExample2

```
import java.net.MalformedURLException;
import java.util.Map;
import javax.xml.ws.BindingProvider;
// Service Imports
import mepapi.com.sap.me.productdefinition.FindOperationConfigurationFault;
import mepapi.com.sap.me.productdefinition.OperationConfigurationServiceWS;
import mepapi.com.sap.me.productdefinition.OperationConfigurationServiceWSService;
// Operation Imports
import com.sap.me.productdefinition.OperationExtendedConfiguration;
import com.sap.me.productdefinition.OperationSearchRequest;
import com.sap.me.productdefinition.OperationSearchResult;
public class PAPIExample2 {

    /*
     * Helper method to log results to the console.
     */
    private static void log(String message) {
        System.out.println(message);
    }

    /**
     * @param args
     */
    public static void main(String[] args)
    {

        // URL wsdlLocation = getWsdl(); // Get the WSDL
        // Change the values here to match your environment.
        String site = "TEST"; // The SAP ME Site
        String operation = "OP1"; // The SAP ME Operation you wish to query

        String user = "SITE_ADMIN"; // The Web service user account
        String password = "[Your password here]"; // The Web service password

        // Set the request for the OperationSearchRequest
        OperationSearchRequest request = new OperationSearchRequest();
        request.setOperation(operation);


        try
        {
            // Create an instance of the Service
           OperationConfigurationServiceWSService service = new OperationConfigurationServiceWSService();
           // Get a reference to the Proxy
           OperationConfigurationServiceWS proxy = service.getOperationConfigurationServiceWSPort();
            // Set the security for Basic Authentication requirements (user/password)
            Map<String,Object> requestContext = ((BindingProvider)proxy).getRequestContext();
            requestContext.put(BindingProvider.USERNAME_PROPERTY, user);
            requestContext.put(BindingProvider.PASSWORD_PROPERTY, password);

            // Send the findOperationConfiguration request
            OperationSearchResult response;
            try {
                response = proxy.findOperationConfiguration(site, request);
                log("Find successful. Details:\n");
                // Response received
                if (response != null)
                {
                    // Iterate through the response and display the results
                    for(OperationExtendedConfiguration c: response.getOperationList())
                    {
                        log("Ref: " +c.getRef());
                        log("Operation: " + c.getOperation());
                        log("Revision: " +c.getRevision());
                        log("Current Revision: " + c.isCurrentRevision().toString());
                    }

                } else
                // Response was null
                {
                    log("Response was null.");
```

```
                }
            } catch (FindOperationConfigurationFault e) {

                // Log Error
                log(e.getMessage());
            }


                log("Done");
        // Catch a Malformed URL Exception
        } catch (MalformedURLException murle)
        {
            log("Exception: " + murle.getMessage());
        }
    }
}
```