Copyright

©Copyright 2008 SAP AG. All rights reserved.

No part of this description of functions may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft®, WINDOWS®, and EXCEL®, NT® and SQL-Server® are registered trademarks of Microsoft Corporation.

IBM®, OS/2®, DB2/6000®, AIX®, OS/400® and AS/400® are registered trademarks of IBM Corporation.

OSF/Motif® is a registered trademark of Open Software Foundation.

ORACLE®, is a registered trademark of ORACLE Corporation, California, USA.

webMethods® is a registered trademark of webMethods Incorporated, Virginia, USA.

INFORMIX®-OnLine for SAP is a registered trademark of Informix Software Incorporated.

UNIX®, and X/Open® are registered trademarks of SCO Santa Cruz Operation.

SAP®, R/2®, R/3®, R/2®, ABAP/4®, SAPaccess®, SAPmail®, SAPoffice®, SAP-EDI®, SAP Business Workflow®, SAP Early Watch®, SAP Archive Link®, R/3 Retail®, ALE/WEB®, SAPTRONIC® are registered trademarks of SAP AG.

All rights reserved.
Contents

Chapter 1. Introduction ......................................................... 19
  Welcome! ........................................................................... 20
  Typographical Conventions ............................................... 20
  Program Code Conventions ............................................... 21
  Related Documentation .................................................... 22
  Viewing this Document .................................................... 23
  Printing this Guide .......................................................... 23

Chapter 2. Getting Started with Developer .............................. 25
  What Is Developer? .......................................................... 26
  Before You Use Developer ................................................ 26
  Starting Developer ........................................................... 26
  The Developer Window ...................................................... 28
    The Service Browser ....................................................... 29
      Service Browser Icons ................................................ 29
      Refreshing the Contents of the Service Browser ................. 31
    The Editor ...................................................................... 31
      Supported Data Types .................................................. 33
  Basic Operations ............................................................. 34
    Working in the Developer Window .................................... 34
    Multiple Areas and Tabs .................................................. 35
    Resizing Areas in the Developer Window ......................... 35
    Creating New Elements .................................................. 36
    Locking and Unlocking Elements ..................................... 36
    Moving and Copying Elements ........................................ 36
    Renaming and Deleting Elements ..................................... 37
    Saving Your Work .......................................................... 40
    Switching between Loaded Elements ................................ 40
    Reverting Changes ......................................................... 42
    Handling Deprecated Elements ....................................... 43
  Opening, Closing, and Restoring Sessions ............................ 47
    Restoring a Session on a Server ....................................... 48
    Notification of Server Shutdown ..................................... 48
    Changing Your Password ................................................. 48
    Password Requirements ................................................ 49
    Caching Elements .......................................................... 50
    Clearing the Developer Cache .......................................... 50
Chapter 3. Working with Packages .................................................. 53
   What Is a Package? ................................................................. 54
   Packages Installed with SAP BC Server ........................................ 54
   Package Management ............................................................ 55
      Creating a Package ............................................................ 57
      Documenting a Package ...................................................... 58
      Reloading a Package ......................................................... 59
      Deleting a Package .......................................................... 59
      Exporting a Package or Partial Package ................................. 60
      Assigning a Version Number for a Package ............................ 61
      Viewing the Patch History for a Package ............................... 62
      Identifying Package Dependencies ..................................... 63
      Removing Package Dependencies .................................... 64
   Assigning Startup, Shutdown, and Replication Services ............... 65
      What Is a Startup Service? ................................................ 65
      What Is a Shutdown Service? ............................................. 66
      What Is a Replication Service? .......................................... 66
      Guidelines for Assigning Startup, Shutdown, and Replication Services .................................................. 66
   Identifying Startup, Shutdown, and Replication Services ........... 67
   Removing Startup, Shutdown, and Replication Services .............. 68

Chapter 4. Building Flow Services .................................................. 69
   Basic Concepts ........................................................................ 70
      What Is a Flow Service? ...................................................... 70
      What Is a Flow Step? ........................................................ 71
      What Is the Pipeline? ......................................................... 73
      What Are Input and Output Parameters? ................................ 74
   A Process Overview ............................................................... 75
   Creating a New Flow Service .................................................. 76
      Package and Folder Requirements ...................................... 76
      Using the Default Logic Options ........................................ 76
      Creating an SAP Inbound Map .............................................. 77
      Generating Elements via SAP Lookup .................................. 78
      Inserting Flow Steps ......................................................... 81
   Declaring Input and Output Parameters for a Service ............... 81
      Specifying Input Parameters ............................................. 82
      Specifying Output Parameters ....................................... 84
      Completing the Input/Output Tab ...................................... 84
      Specifying Run-Time Parameters .................................... 87
## Contents

- Maintaining the State of a Service .................................................. 87
- Configuring a Service’s Use of Cache .............................................. 88
  - Types of Services to Cache ...................................................... 88
  - Controlling a Service’s Use of Cache .......................................... 89
  - Specifying the Duration of a Cached Result .................................. 90
  - Using the Prefetch Option ......................................................... 90
- Assigning an Output Template to a Service ....................................... 91
- Managing Security for Services .................................................... 93
  - Assigning ACLs to Folders and Services ...................................... 94
    - Assigning an ACL to a Folder ............................................... 95
    - Assigning an ACL to a Service ............................................. 95
    - Requiring ACL Checking for Internal Invokes ............................ 96
    - Removing ACLs from a Folder or Service .................................. 97
    - The As-User Property in SAP BC Developer 3.x ......................... 97
  - Assigning Universal Names to Services ...................................... 98
  - Printing a Flow Service ........................................................ 99

## Chapter 5. Inserting Flow Steps ...................................................... 101

- Inserting and Moving Flow Steps ............................................... 102
  - Changing the Position of a Flow Step ........................................ 104
  - Changing the Level of a Flow Step .......................................... 104
  - Setting the Properties of a Flow Step ...................................... 105
- Invoking Services ........................................................................... 107
  - Specifying the Service Property .............................................. 107
  - Invoking a Built-In Service .................................................... 108
  - Invoking a Service on Another SAP BC Server ............................ 108
  - Building an INVOKE Step ....................................................... 108
  - Finding an Invoked Service ..................................................... 110
- The BRANCH Step ........................................................................... 110
  - Branching on a Switch Value .................................................... 110
    - Specifying the Switch Variable .............................................. 111
    - Specifying the Label Value .................................................. 111
  - Branching on an Expression .................................................... 113
  - Specifying a Default Step ....................................................... 114
  - Branching on Null and Empty Values ....................................... 115
  - Using SEQUENCE as the Target of a BRANCH ............................. 115
  - Building a BRANCH Step ........................................................ 116
- The REPEAT Step ............................................................................ 118
  - Specifying the REPEAT Condition ............................................. 119
  - Setting the REPEAT Counter .................................................... 120
  - When Does REPEAT Fail? ........................................................ 120
## Contents

Using REPEAT to Retry a Failed Step ................................................................. 120
Using REPEAT to Retry a Successful Step ......................................................... 123
The SEQUENCE Step ......................................................................................... 125
Using SEQUENCE to Specify an Exit Condition .................................................. 125
The LOOP Step .................................................................................................... 126
Specifying the Input Array ................................................................................. 127
Collecting Output from a LOOP Step ................................................................. 128
Building a LOOP Step ......................................................................................... 129
The EXIT Step ...................................................................................................... 130
The MAP Step ...................................................................................................... 132

### Chapter 6. Using Flow Diagram View ............................................................. 133

What Is Flow Diagram View? ............................................................................... 134
When Should You Use Flow Diagram View? ....................................................... 135
What Does a Flow Service Look Like in Diagram View? .................................... 136
Switching to Flow Diagram View ......................................................................... 140
Building a Flow Service in Diagram View .......................................................... 140
Inserting a Flow Step ........................................................................................... 140
Inserting a Child Step into a BRANCH Step ....................................................... 142
  Inserting an Step into an Existing Target .......................................................... 143
Changing the Order of Steps in a Flow Service ................................................... 147
Changing the Layout of a Flow Service ............................................................... 147
Resetting the Flow Service Layout ..................................................................... 148

### Chapter 7. Mapping Data in a Flow Service ................................................... 149

What is the Pipeline Editor? .................................................................................. 150
Pipeline Editor for an INVOKE Step ..................................................................... 151
Pipeline Editor for a MAP Step .......................................................................... 152
Pipeline Modifiers ............................................................................................... 153
Printing the Pipeline Editor ................................................................................ 154
Basic Mapping Tasks ............................................................................................ 154
Mapping Variables ............................................................................................... 155
  What Happens When SAP BC Server Executes a Map Between Variables at Run Time? 160
  Mapping to Record Variables ............................................................................ 163
  Mapping Variables of Different Data Types ....................................................... 163
  Mapping to and from Array Variables .............................................................. 165
  Deleting Links Between Variables ..................................................................... 169
  Applying Conditions to Maps Between Variables ............................................. 169
Assigning Values to Pipeline Variables .............................................................. 172
  Assigning a Default Value to a Variable .......................................................... 173
  Referencing Other Variables ............................................................................ 173
What Is an XML Document Validated Against? ........................................ 218
Performing Pipeline Validation ............................................................. 222
Performing Input/Output Validation ......................................................... 226
  Specifying Input/Output Validation via the Input/Output Tab .................... 227
  Specifying Input/Output Validation via the INVOKE Step ......................... 228
Performing Record and Node Validation .................................................. 229
Validation Errors and Exceptions ............................................................ 233
  Validation Exceptions ......................................................................... 233
  Running Out of Memory During Validation ......................................... 234

Chapter 10. Managing Service and Record Dependencies ......................... 235
  Basic Concepts ................................................................................. 236
  What Is an SAP BC Element? .............................................................. 236
  What Is a Dependent? ....................................................................... 236
  What Is a Reference? ......................................................................... 237
  What Is a Pipeline Reference? ............................................................. 237
  Finding SAP BC Elements in the Service Browser .................................. 238
  Finding Dependents .......................................................................... 240
  Checking Dependencies when Renaming SAP BC Elements .................... 242
  Finding References ........................................................................... 242
  Inspecting Pipeline References .......................................................... 244

Chapter 11. Testing and Debugging Services ............................................ 247
  Testing and Debugging ....................................................................... 248
  Testing Services ................................................................................. 248
  Testing Services from Developer .......................................................... 249
    Entering Input for a Service .............................................................. 250
    Saving Input Values to a File ............................................................ 251
    Loading Input Values from a File ....................................................... 252
  Viewing the Results of the Service ....................................................... 253
    Copying Variables From the Results Tab .......................................... 255
  Run-Time Exceptions ......................................................................... 255
    The Call Stack ............................................................................... 256
    The Pipeline Dump ......................................................................... 257
  Testing Services from a Browser .......................................................... 257
  Testing Services that Expect XML Documents as Input ................................ 258
  Working in Debug Mode .................................................................... 259
    Entering Debug Mode ...................................................................... 260
    Combining the Step and Trace Commands in Debug Mode .................. 261
    Resetting Debug Mode .................................................................... 261
    Using the Trace Tools ..................................................................... 262
## Contents

**Chapter 11. Debugging**

- Tracing into a Child Flow .................................................. 263
- Using the Step Tools ......................................................... 264
- Stepping through a Child Flow ........................................... 265
- Using the Step Tools with a MAP Step ................................. 266

- Setting Breakpoints ....................................................... 266
- What Happens When a Breakpoint is Encountered? .................. 267
- Setting Breakpoints on Transformers .................................. 268
- Viewing a List of Breakpoints ........................................... 269

- Disabling Flow Steps, Transformers, and Conditions ............... 269
  - Disabling Flow Steps .................................................... 269
  - Disabling Transformers ............................................... 271
  - Disabling a Condition Placed on a Map Between Variables ....... 273

- Modifying the Current Pipeline ......................................... 274

- Saving and Restoring the Pipeline .................................... 275
  - Saving the Results ..................................................... 275
  - Saving the Contents of the Results Tab ............................. 276
  - Saving the Pipeline at Run Time .................................... 277

- Restoring the Pipeline .................................................... 278
  - Loading a Saved Pipeline into the Results Tab ..................... 278
  - Loading a Saved Pipeline at Run Time ............................... 279

- Other Debugging Techniques ............................................ 280
  - Using the Server’s Debug Facility .................................... 280
  - The Contents of server.log .......................................... 280
  - Server Debug Levels .................................................. 281
  - Writing Information to server.log .................................. 281
    - Writing an Arbitrary Message to the Log ......................... 282
    - Dumping the Pipeline to the Log ................................ 283

**Chapter 12. Building Coded Services** ................................. 287

- Basic Concepts .............................................................. 288
  - The IData Object ....................................................... 288
    - Services Take IData Objects as Input and Return IData as Output ........................................ 288
    - Getting and Setting Elements in an IData object ................ 289
    - Creating IData objects .............................................. 289
  - The Values Object ..................................................... 290

- Building Services Using Java ......................................... 290
- How Java Services Are Organized on the Server ....................... 290

- Building Java Services with Developer ................................ 291
  - Using the Developer IDE ............................................... 292
    - The Source Tab ...................................................... 292
    - The Shared Tab ...................................................... 293
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating a Java Service with Developer's IDE</td>
<td>295</td>
</tr>
<tr>
<td>Generating Java Code from Service Input and Output Parameters</td>
<td>296</td>
</tr>
<tr>
<td>Setting Run-Time Options for a Java Service</td>
<td>297</td>
</tr>
<tr>
<td>Building Java Services with Your Own IDE</td>
<td>298</td>
</tr>
<tr>
<td>The Namespace Directory</td>
<td>298</td>
</tr>
<tr>
<td>The Source Code Directory</td>
<td>299</td>
</tr>
<tr>
<td>Writing the Source Code for a Service</td>
<td>299</td>
</tr>
<tr>
<td>Using the SAP BC API</td>
<td>300</td>
</tr>
<tr>
<td>The Basic Stages</td>
<td>301</td>
</tr>
<tr>
<td>Commenting Code for the SAP BC Server</td>
<td>301</td>
</tr>
<tr>
<td>Using the jcode Utility</td>
<td>302</td>
</tr>
<tr>
<td>Make Mode</td>
<td>302</td>
</tr>
<tr>
<td>Fragment Mode</td>
<td>303</td>
</tr>
<tr>
<td>Composite Mode</td>
<td>303</td>
</tr>
<tr>
<td>Other jcode Commands</td>
<td>304</td>
</tr>
<tr>
<td>Debugging a Java Service</td>
<td>304</td>
</tr>
<tr>
<td>Preparing Debugging</td>
<td>304</td>
</tr>
<tr>
<td>Running the Debugger</td>
<td>305</td>
</tr>
<tr>
<td>Building Services Using C/C++</td>
<td>306</td>
</tr>
<tr>
<td>Generating Files for a C/C++ Service</td>
<td>307</td>
</tr>
<tr>
<td>The Java Code for a C Service</td>
<td>307</td>
</tr>
<tr>
<td>Building the C/C++ Source Code</td>
<td>309</td>
</tr>
<tr>
<td>Writing the Service</td>
<td>310</td>
</tr>
<tr>
<td>Compiling the Service</td>
<td>310</td>
</tr>
<tr>
<td>Building Services Using COM</td>
<td>310</td>
</tr>
<tr>
<td>Requirements</td>
<td>311</td>
</tr>
<tr>
<td>Invoking Methods from Existing COM and DCOM Objects</td>
<td>311</td>
</tr>
<tr>
<td>Creating the Object</td>
<td>311</td>
</tr>
<tr>
<td>Invoking the Object</td>
<td>312</td>
</tr>
<tr>
<td>Writing a Visual Basic Service</td>
<td>312</td>
</tr>
<tr>
<td>Compiling a Visual Basic Service</td>
<td>313</td>
</tr>
<tr>
<td>Invoking a Visual Basic Service</td>
<td>313</td>
</tr>
<tr>
<td>Invoking a VB Service Using Late Binding</td>
<td>313</td>
</tr>
<tr>
<td>Invoking a VB Service Using Early Binding</td>
<td>315</td>
</tr>
</tbody>
</table>

**Chapter 13. Creating Client Code**  .................................................. 319

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Concepts</td>
<td>320</td>
</tr>
<tr>
<td>Building a Java Client</td>
<td>320</td>
</tr>
<tr>
<td>Assumptions</td>
<td>320</td>
</tr>
<tr>
<td>Third-Party Libraries in client.jar</td>
<td>321</td>
</tr>
<tr>
<td>Limitations</td>
<td>321</td>
</tr>
</tbody>
</table>
Chapter 14. Working with WSDL ................................. 335

Overview .......................................................... 336
What Is WSDL? .................................................... 336
What Does a WSDL Document Look Like? ....................... 336
The WSDL Namespace Declaration .............................. 339
Generating a WSDL Document for a Service ................. 340
Generating a WSDL Document that Uses the SOAP RPC Protocol ......................................................... 341
What Files are Generated? ........................................ 344
Generating a WSDL that Uses the SOAP Message Protocol ................................................................. 352
Service Requirements for Using the Default SOAP Processor ............................................................. 353
Chapter 15. Passing XML Data to a Service ........................................... 371
Sending XML Documents to SAP BC Server .................................................. 372
Submitting an XML Document in a String Variable ......................................... 372
Submitting an XML Document in $xmldata ....................................................... 373
Posting an XML Document via HTTP .............................................................. 375
Submitting an XML Document via FTP ............................................................ 376
Building a Client that Sends an XML Document to a Service ............................... 377
FTPing a File From a SAP BC Server ............................................................... 377
Getting Results from an FTPed Document ....................................................... 377
Submitting an XML Document via Email ......................................................... 378
Requirements for Sending an XML Document via Email ..................................... 378
Getting Results from an Emailed Document ..................................................... 379

Chapter 16. Using the Load and Query Services ............................................. 381
What Are the Load and Query Services? ......................................................... 382
Basic Concepts ............................................................................................... 382
What Is Parsing? ............................................................................................... 382
What Is a Document Object? ............................................................................ 383
What Is a Node? ............................................................................................... 383
What Is a Query? ............................................................................................. 383
Using the loadDocument Service .................................................................... 383
Adding loadDocument to a Flow Service .......................................................... 384
Input and Output Values for loadDocument ....................................................... 384
Inputs for loadDocument .................................................................................. 385
  The url variable ............................................................................................ 385
  The method variable ..................................................................................... 386
  The auth variable ......................................................................................... 387
  The data variable .......................................................................................... 388
  The headers variable .................................................................................... 393
  The encoding variable .................................................................................. 395
  The expandDTD variable ............................................................................. 396
  The isXML variable ...................................................................................... 396
  The loadAs variable ..................................................................................... 397
## Contents

The failOnHTTPError variable ................................................. 398
Output Values from loadDocument .......................................... 398
Error Handling With loadDocument ......................................... 399
Using the queryDocument Service ........................................... 400
Passing a Node to queryDocument .......................................... 400
Addressing Information in a Node .......................................... 401
Working with a Sample Document .......................................... 402
  The Document View Tab .................................................... 403
  The Sample View Tab ...................................................... 403
Adding queryDocument to a Flow Service ................................. 404
Specifying a Query ............................................................ 406
  Specifying a Query Manually ............................................. 407
  Specifying a Query by Pointing to a Node ............................. 409
Using WQL and XQL ............................................................ 410
  Using WQL in a Query ..................................................... 410
  Using XQL in a Query .................................................... 419
  Quick Reference: WQL vs. XQL .......................................... 426
Output from queryDocument .................................................. 427
Handling Errors from queryDocument ....................................... 427
Load and Query Shortcuts .................................................... 427
  Generating Default Variables ............................................ 427
  Creating a Load and Query Service from a URL ...................... 428
  Using the Browser Recorder .............................................. 430

### Chapter 17. Accessing Databases with Services ....................... 433
The Types of Services You Can Create .................................... 434
Deciding What Type of Service to Create ................................ 434
Creating Database Flow Services .......................................... 435
  Generating a Service from a SQL Statement ......................... 436
  Specifying a Dynamic SQL Statement ................................ 436
  Steps for Generating the Flow Service ............................... 438
  Generating a Service from a Table .................................... 439
  Restricting the List of Database Tables ............................. 439
  Steps for Generating the Flow Service ............................... 440
  Output from the Flow Service ......................................... 442
Creating Database Services with Java, C/C++, or VB ................... 442
  Built-in Database Services .............................................. 442
  Testing SQL Statements .................................................. 443
  Error Handling ............................................................. 444
Creating Clients that Access Databases .................................. 444
  Invoking a Database Service from a Browser-based Client ......... 444
Chapter 18. Using Guaranteed Delivery ........................................... 449
What Is Guaranteed Delivery? ........................................... 450
Indicating You Want to Use Guaranteed Delivery ......................... 450
How Transactions Are Managed ....................................... 450
  Customizing the Job Manager ...................................... 451
Identifying Transactions ............................................ 452
Specifying How Long Transactions are Active ......................... 452
Handling Failures ...................................................... 453
Creating a Java Client that Uses Guaranteed Delivery ............... 454
  Sample Code (Synchronous Request) ................................ 454
  Sample Code (Asynchronous Request) ................................ 457
Creating a Flow Service that Uses Guaranteed Delivery ............. 458
  Sample Flow (Synchronous Request) ................................ 459
  Sample Flow (Asynchronous Request) ................................ 460

Chapter 19. Working with MIME and S/MIME Messages ............... 463
Overview ................................................................. 464
What Is MIME? ......................................................... 464
  Basic Structure of a MIME Message ................................ 464
    Header Fields ...................................................... 465
    The Body ......................................................... 465
  Multipart MIME Messages ......................................... 466
What Is S/MIME? ....................................................... 467
Digital Certificates ................................................... 467
Digital Signatures ..................................................... 468
  Explicit and Implicit Signatures .................................. 468
Encryption .............................................................. 470
The MIME and S/MIME Services .................................... 472
  Services Used to Construct MIME and S/MIME Messages .......... 472
  Services Used to Extract Data from MIME and S/MIME Messages ... 472
MIME Messages, MIME Entities and MIME Objects .................. 473
Building MIME and S/MIME Messages ................................ 473
Creating a MIME Message ............................................ 473
  How to Create a MIME Message ................................... 474
    Example—Creating a Single-Part MIME Message ................. 478
    Example—Creating a Multipart MIME Message .................. 480
Signing a MIME Message ............................................. 483
### Contents

- **How to Create a Signed S/MIME Message** ................................................................. 483
  - Example—Signing a MIME Message ........................................................................ 484
- **Encrypting a MIME Message** .................................................................................. 487
  - How to Create an Encrypted S/MIME Message .................................................... 487
  - Example—Encrypting a MIME Message .................................................................. 487
- **Signing and Encrypting a MIME Message** .............................................................. 489
  - Example—Signing and Encrypting a MIME Message ............................................ 489
- **Extracting Data from MIME and S/MIME Messages** ............................................ 492
  - Extracting the Payload from a MIME Message .................................................. 492
  - How to Extract the Payload from a MIME Message ............................................ 492
  - Example—Extracting One Part from a Multipart MIME Message ......................... 494
  - Example—Extracting All Parts from a Multipart MIME Message ......................... 495
  - Extracting the Payload from a Signed MIME Message ....................................... 496
  - How Do You Know Whether the Message Is Signed? ........................................ 496
  - Working with InputStreams .................................................................................... 496
  - What Happens when the Signature is Processed? .............................................. 497
  - Error Codes and Messages .................................................................................... 497
  - How to Extract the Payload from a Signed S/MIME Message ............................... 498
  - Example—Extracting Content from a Signed S/MIME Message ......................... 499
  - Extracting the Payload from an Encrypted MIME Message ................................. 500
  - How Do You Know Whether the Message Is Encrypted? .................................... 500
  - How to Extract the Payload from an Encrypted S/MIME Message ....................... 500
  - Example—Extracting Content from an Encrypted S/MIME Message ..................... 501
  - Extracting Data from a Signed and Encrypted MIME Message ............................ 502
  - Example—Extracting Content from a Signed and Encrypted S/MIME Message ...... 503

### Chapter 20. Using WebTap .................................................................................. 505

- **What Is WebTap?** .................................................................................................. 506
- **Components of a WebTap Application** ............................................................... 506
- **How Does a WebTap Service Work?** ................................................................ 507
- **Guidelines for Using WebTap** ............................................................................. 507
- **Creating a WebTap Service** ................................................................................ 508
- **Defining WebTap Rules** ..................................................................................... 510
- **Creating Before and After Services** .................................................................... 511
  - A Simple Rerouting Example .............................................................................. 516
  - Debugging a WebTap Service .............................................................................. 518
- **Creating a Customized WebTap Service** ............................................................. 518
- **Invoking a WebTap Service from a Browser** ...................................................... 520

### Chapter 21. Subscribing to Events ...................................................................... 523

- **The Event Manager** ........................................................................................... 524
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>What Are Event Handlers?</td>
<td>525</td>
</tr>
<tr>
<td>What Happens When an Event Occurs?</td>
<td>525</td>
</tr>
<tr>
<td>Managing Event Subscriptions</td>
<td>526</td>
</tr>
<tr>
<td>Subscribing to an Event</td>
<td>526</td>
</tr>
<tr>
<td>Creating Event Filters</td>
<td>529</td>
</tr>
<tr>
<td>Creating Event Filters for Services</td>
<td>531</td>
</tr>
<tr>
<td>Viewing and Editing Event Subscriptions</td>
<td>531</td>
</tr>
<tr>
<td>Suspending Event Subscriptions</td>
<td>532</td>
</tr>
<tr>
<td>Deleting an Event Subscription</td>
<td>532</td>
</tr>
<tr>
<td>Building an Event Handler</td>
<td>533</td>
</tr>
<tr>
<td>Sample Event Handler</td>
<td>533</td>
</tr>
<tr>
<td>Working with Alarm Events</td>
<td>535</td>
</tr>
<tr>
<td>Building Handlers for Alarm Events</td>
<td>536</td>
</tr>
<tr>
<td>Working with Audit Events</td>
<td>536</td>
</tr>
<tr>
<td>Setting Audit Levels</td>
<td>537</td>
</tr>
<tr>
<td>Building Handlers for Audit Events</td>
<td>538</td>
</tr>
<tr>
<td>Default Audit Event Handler</td>
<td>540</td>
</tr>
<tr>
<td>Working with Exception Events</td>
<td>540</td>
</tr>
<tr>
<td>Building Handlers for Exception Events</td>
<td>540</td>
</tr>
<tr>
<td>Default Exception Event Handler</td>
<td>541</td>
</tr>
<tr>
<td>Working with Guaranteed Delivery Events</td>
<td>542</td>
</tr>
<tr>
<td>Guaranteed Delivery Events and Transaction Events</td>
<td>542</td>
</tr>
<tr>
<td>Building Handlers for Guaranteed Delivery Start Events</td>
<td>543</td>
</tr>
<tr>
<td>Building Handlers for Guaranteed Delivery End Events</td>
<td>544</td>
</tr>
<tr>
<td>Default GD Start and GD End Event Handler</td>
<td>544</td>
</tr>
<tr>
<td>Working with Port Status Events</td>
<td>545</td>
</tr>
<tr>
<td>Building Handlers for Port Status Events</td>
<td>545</td>
</tr>
<tr>
<td>Working with Replication Events</td>
<td>546</td>
</tr>
<tr>
<td>Building Handlers for Replication Events</td>
<td>546</td>
</tr>
<tr>
<td>Working with Session Events</td>
<td>546</td>
</tr>
<tr>
<td>Building Handlers for Session Start Events</td>
<td>547</td>
</tr>
<tr>
<td>Default Session Start Event Handler</td>
<td>547</td>
</tr>
<tr>
<td>Building Handlers for Session End Events</td>
<td>548</td>
</tr>
<tr>
<td>The Default Session End Event Handler</td>
<td>548</td>
</tr>
<tr>
<td>Building Handlers for Session Expire Events</td>
<td>548</td>
</tr>
<tr>
<td>Default Session Expire Event Handler</td>
<td>549</td>
</tr>
<tr>
<td>Working with Stat Events</td>
<td>549</td>
</tr>
<tr>
<td>Building Handlers for Stat Events</td>
<td>549</td>
</tr>
<tr>
<td>Default Stat Event Handler</td>
<td>550</td>
</tr>
<tr>
<td>Working with Transaction Events</td>
<td>550</td>
</tr>
<tr>
<td>Contents</td>
<td>551</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>Building Handlers for Transaction Start Events</td>
<td></td>
</tr>
<tr>
<td>Building Handlers for Transaction End Events</td>
<td></td>
</tr>
<tr>
<td>Default Tx Start and Tx End Event Handler</td>
<td></td>
</tr>
<tr>
<td>Appendix A. SAP BC Flow Steps</td>
<td>553</td>
</tr>
<tr>
<td>BRANCH</td>
<td></td>
</tr>
<tr>
<td>Branching on a Switch Value</td>
<td>554</td>
</tr>
<tr>
<td>Branching on Expressions</td>
<td>554</td>
</tr>
<tr>
<td>Properties</td>
<td>555</td>
</tr>
<tr>
<td>Conditions That Will Cause a BRANCH Step to Fail</td>
<td>556</td>
</tr>
<tr>
<td>EXIT</td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td>557</td>
</tr>
<tr>
<td>Examples of When to Use</td>
<td>558</td>
</tr>
<tr>
<td>INVOKE</td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td>559</td>
</tr>
<tr>
<td>Conditions That Will Cause an INVOKE Step to Fail</td>
<td>560</td>
</tr>
<tr>
<td>LOOP</td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td>561</td>
</tr>
<tr>
<td>Conditions That Will Cause a LOOP Step to Fail</td>
<td>562</td>
</tr>
<tr>
<td>MAP</td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td>563</td>
</tr>
<tr>
<td>Example of When to Use</td>
<td>564</td>
</tr>
<tr>
<td>REPEAT</td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td>565</td>
</tr>
<tr>
<td>When Does REPEAT Fail?</td>
<td>566</td>
</tr>
<tr>
<td>Examples of When to Use</td>
<td>567</td>
</tr>
<tr>
<td>SEQUENCE</td>
<td></td>
</tr>
<tr>
<td>Properties</td>
<td>568</td>
</tr>
<tr>
<td>Conditions That Will Cause the SEQUENCE Step to Fail</td>
<td>569</td>
</tr>
<tr>
<td>Appendix B. webMethods Query Language</td>
<td>571</td>
</tr>
<tr>
<td>Overview</td>
<td></td>
</tr>
<tr>
<td>Object References</td>
<td>572</td>
</tr>
<tr>
<td>Sibling Operators</td>
<td>574</td>
</tr>
<tr>
<td>Object Properties</td>
<td>575</td>
</tr>
<tr>
<td>Property Masking</td>
<td>576</td>
</tr>
<tr>
<td>Appendix C. Supported Data Types</td>
<td>577</td>
</tr>
<tr>
<td>Data Types</td>
<td></td>
</tr>
<tr>
<td>How SAP BC Server Handles Objects and Object Lists</td>
<td>579</td>
</tr>
<tr>
<td>How the SAP BC Developer Supports Tables</td>
<td>579</td>
</tr>
<tr>
<td>Contents</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>Values Objects and IData .................................................. 580</td>
<td></td>
</tr>
<tr>
<td>Mapping Data Types .......................................................... 580</td>
<td></td>
</tr>
<tr>
<td>Default Pipeline Rules for Mapping to and from Array Variables ......... 580</td>
<td></td>
</tr>
<tr>
<td>Appendix D. Regular Expressions ............................................ 585</td>
<td></td>
</tr>
<tr>
<td>What Is a Regular Expression? ................................................ 586</td>
<td></td>
</tr>
<tr>
<td>Using a Regular Expression in a Mask ....................................... 586</td>
<td></td>
</tr>
<tr>
<td>Regular Expression Operators ............................................... 587</td>
<td></td>
</tr>
<tr>
<td>Appendix E. Conditional Expressions .................................... 593</td>
<td></td>
</tr>
<tr>
<td>Overview ................................................................. 594</td>
<td></td>
</tr>
<tr>
<td>Syntax ................................................................. 594</td>
<td></td>
</tr>
<tr>
<td>Checking for Variable Existence ......................................... 596</td>
<td></td>
</tr>
<tr>
<td>Operators ............................................................. 597</td>
<td></td>
</tr>
<tr>
<td>Relational Operators .................................................. 597</td>
<td></td>
</tr>
<tr>
<td>Logical Operators .................................................... 599</td>
<td></td>
</tr>
<tr>
<td>Precedence .......................................................... 600</td>
<td></td>
</tr>
<tr>
<td>Addressing Pipeline Variables ............................................ 601</td>
<td></td>
</tr>
<tr>
<td>Appendix F. jcode tags .................................................... 603</td>
<td></td>
</tr>
<tr>
<td>jcode Template .......................................................... 604</td>
<td></td>
</tr>
<tr>
<td>jcode Examples .......................................................... 605</td>
<td></td>
</tr>
<tr>
<td>Sample Code - IData .................................................... 605</td>
<td></td>
</tr>
<tr>
<td>Sample Code - Values .................................................. 609</td>
<td></td>
</tr>
<tr>
<td>Appendix G. Validation Content Constraints .......................... 611</td>
<td></td>
</tr>
<tr>
<td>Overview ............................................................. 612</td>
<td></td>
</tr>
<tr>
<td>Content Types .......................................................... 612</td>
<td></td>
</tr>
<tr>
<td>Constraining Facets .................................................... 623</td>
<td></td>
</tr>
<tr>
<td>Appendix H. Validation Errors and Exceptions .......................... 627</td>
<td></td>
</tr>
<tr>
<td>Overview ............................................................. 628</td>
<td></td>
</tr>
<tr>
<td>Validation Errors .................................................... 628</td>
<td></td>
</tr>
<tr>
<td>Validation Exceptions .................................................. 639</td>
<td></td>
</tr>
<tr>
<td>SAP BC Schema Generation Errors and Warnings ......................... 643</td>
<td></td>
</tr>
<tr>
<td>Appendix I. WSDL Errors and Warnings ................................ 647</td>
<td></td>
</tr>
<tr>
<td>Overview ............................................................. 648</td>
<td></td>
</tr>
<tr>
<td>WSDL Related Errors and Warnings ..................................... 649</td>
<td></td>
</tr>
<tr>
<td>Index ............................................................. 661</td>
<td></td>
</tr>
</tbody>
</table>
Welcome!

This guide shows you how to create services using SAP BC Developer®. It contains information for developers who want to build services using the webMethods flow language or a programming language such as Java, C/C++, or Visual Basic. It also contains information about creating client applications, working with output templates, and accessing databases through the SAP BC platform.

To use this guide effectively, you should know how to program in Java, C/C++, and/or Visual Basic if you will be creating services in those languages.

Typographical Conventions

This document uses the following typographical conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedures are designated by a blue box in the left column. Procedures are presented as a series of numbered steps.</td>
<td>1 On the <strong>Activity</strong> menu, click <strong>File</strong>.</td>
</tr>
<tr>
<td>Terms that identify elements, options, selections, and commands on the screen are shown in bold.</td>
<td>The <strong>Service</strong> field on the <strong>Properties</strong> tab specifies the name of the requested service.</td>
</tr>
<tr>
<td>Characters that you must type exactly are shown in a typewriter font.</td>
<td><strong>Type</strong>: <code>setup</code> and then press ENTER.</td>
</tr>
<tr>
<td>Variable information that you must type based on your specific situation or environment is shown in italics.</td>
<td><strong>Type</strong>: <code>&lt;sapbc&gt;/setup</code> and then press ENTER.</td>
</tr>
<tr>
<td>Keyboard keys are shown in uppercase.</td>
<td>Press ENTER; then press TAB.</td>
</tr>
<tr>
<td>Keys that you must press simultaneously are joined with the “+” symbol.</td>
<td>Press CTRL+ALT+M.</td>
</tr>
<tr>
<td>Directory paths are shown with the “/” directory delimiter unless the subject is UNIX-specific. In these cases, the “/” is used. If you are working in a UNIX environment, substitute a “/” for the “/” shown in the procedures in this book.</td>
<td><code>&lt;sapbc&gt;/server/packages\Default</code></td>
</tr>
</tbody>
</table>
Typographical Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information that you must read before beginning a procedure or that alerts you to negative consequences of certain actions is denoted using this notation.</td>
<td>Important! If the folder is not already open in the Service Browser, open it before you start the following procedure.</td>
</tr>
<tr>
<td>Notes that provide related, but non-critical, information are denoted using this notation.</td>
<td>Note: When you start SAP BC Developer, you are prompted to log on to a SAP BC Server.</td>
</tr>
<tr>
<td>Helpful information such as shortcuts and alternatives.</td>
<td>Tip! You can also use CTRL+C to copy an object.</td>
</tr>
</tbody>
</table>

Program Code Conventions

For programming code and command syntax, this document uses the following typographical conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keywords and values that you must type exactly as printed are shown in typewriter font.</td>
<td>%CoSymbol%</td>
</tr>
<tr>
<td>Variable values or parameters that you must supply are shown in italics.</td>
<td>%VarName%</td>
</tr>
<tr>
<td>Keywords or values that are optional are enclosed in [ ]. Do not type the [ ] symbols in your own code.</td>
<td>%loop LoopVar [null=NullValue]%</td>
</tr>
</tbody>
</table>
# Related Documentation

The following documents are companions to this guide. Some documents are in PDF format and others are in HTML.

<table>
<thead>
<tr>
<th>Refer to this book...</th>
<th>For...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SAP BC Built-In Services Guide</strong></td>
<td>Descriptions of services that are installed on your SAP BC Server. This book is for solution developers. You will find this book at: &lt;sapbc&gt;/developer/doc/SAPBCBuiltInServicesGuide.pdf</td>
</tr>
<tr>
<td><strong>Building Output Templates and DSPs</strong></td>
<td>Information about creating output templates and Dynamic Server Pages (DSPs). This reference is for solution developers. You will find this book at: &lt;sapbc&gt;/developer/doc/SAPBCTemplatesAndDSPs.pdf</td>
</tr>
<tr>
<td><strong>SAP BC Java API Reference</strong></td>
<td>Descriptions of the Java classes you use to create services. This reference is for developers who build services using Java. You will find this book at: &lt;sapbc&gt;/developer/doc/api/Java/index.html</td>
</tr>
<tr>
<td><strong>SAP BC Developer Tutorial</strong></td>
<td>Information that orients you to the SAP BC Developer and shows you how to create a simple application. It includes basic conceptual information about the SAP BC Developer. This book is for users of the SAP BC Developer. You will find this book at: &lt;sapbc&gt;/developer/doc/SAPBCDevTutorial.pdf</td>
</tr>
<tr>
<td>Refer to this book...</td>
<td>For...</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Developer Online Reference</td>
<td>Information about the controls in the Developer application windows and step-by-step procedures describing how to perform tasks with the Developer. You can access the online reference by clicking Help in an application window or dialog box.</td>
</tr>
</tbody>
</table>

**Viewing this Document**

To view this document, which is in PDF format, you must have Acrobat Reader™ 4.0 or later installed on your system. If you have an earlier version of Acrobat Reader, you will receive the following error message when you open this document and Acrobat Reader will not display the images in this document:

*Could not find the ColorSpace named 'Cs8.'*

If you do not have this software or you do not have the correct version, you can download a free copy from:


**Printing this Guide**

To produce a hard copy of this guide, print this document from Acrobat Reader.
Getting Started with Developer

- What Is Developer? ............................................................... 26
- Before You Use Developer .................................................. 26
- Starting Developer .............................................................. 26
- The Developer Window ......................................................... 28
- Basic Operations ............................................................... 34
- Opening, Closing, and Restoring Sessions ............................... 47
- Changing Your Password ..................................................... 48
- Caching Elements ............................................................. 50
- Using Online Help ............................................................ 51
CHAPTER 2  Getting Started with Developer

What Is Developer?

SAP BC Developer is an integrated development environment (IDE) that you use to create services on a SAP BC Server. It provides all of the tools necessary to build and test services, generate stubs for client applications, and create output templates.

Before You Use Developer

Developer builds and edits services directly on a server. To use Developer you must:

- Have access to a SAP BC Server on which you can build and test services.
- Have a user account on that SAP BC Server.
- Belong to a group that is a member of the “Developers” ACL (access control list) on that SAP BC Server.

You will not be able to use Developer unless these requirements are satisfied. If you do not have access to a SAP BC Server or you do not have an appropriate user account or access rights, contact your server administrator.

Starting Developer

Use the following procedure to start Developer on your workstation.

⚠️ Important! Make sure that the server with which you want to use Developer is running. You cannot work with Developer if the server is not running.

⚠️ Important! If you are starting Developer for the first time on a UNIX system, verify that the B2B_ROOT and JAVA_ROOT settings in integrator.sh specify the paths where Developer (B2B_ROOT) and Java Runtime Environment (JAVA_ROOT) reside. If these settings are not correct, update them before starting Developer.
To start Developer

1 Depending on which operating system is running on your workstation, do the following:

<table>
<thead>
<tr>
<th>If you are running...</th>
<th>Do this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>1 On the Start menu, click Programs, and then click SAP BC.</td>
</tr>
<tr>
<td></td>
<td>2 Click SAP BC Developer.</td>
</tr>
<tr>
<td>UNIX</td>
<td>1 Navigate to the directory where you installed Developer.</td>
</tr>
<tr>
<td></td>
<td>2 Run bin/integrator.sh.</td>
</tr>
</tbody>
</table>

Specify the name and port assignment of a server... and enter a user account that has developer privileges.

3 In the Open Session dialog box, complete the following:

<table>
<thead>
<tr>
<th>In this field...</th>
<th>Specify...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server type</td>
<td>The registered type for the server on which you want to open a session. The default type is Integration Server.</td>
</tr>
<tr>
<td>Server</td>
<td>The name and port assignment of the SAP BC Server in ServerName:PortNum format.</td>
</tr>
<tr>
<td>Example rubicon:5555</td>
<td></td>
</tr>
</tbody>
</table>

Note: Servers to which you have successfully logged on in the past are listed in the Server list. You can select a server from this list or type its name and port number.
In this field... | Specify...
---|---
Username | The name of a valid user account on this server. (The user name must be a member of a group belonging to the Developers ACL.)
| Use the exact combination of upper- and lower-case characters with which it was originally defined. SAP BC user names are case-sensitive.

Note: The server is installed with a default user account called “Developer” that has developer privileges.

Password | The password for the user account in Username. Use the exact combination of upper- and lower-case characters with which it was originally defined. SAP BC passwords are case-sensitive.

Note: The default password for the Developer user account is isdev.

Uses secure connection | Whether the session will be opened through HTTP or HTTPS. If you want to open a session on the selected server using the Secure Socket Layer (SSL), select this check box.

Note: If you want to use a proxy you can configure it via Edit -> Preferences -> Proxies. I you get a transport error exception when trying to login, check your proxy settings.

4 Click **OK**.

**The Developer Window**

The Developer window is divided into two areas: the “Service Browser” (on the left) and the “Editor” (on the right).
The Service Browser

The Service Browser displays the contents of packages on the SAP BC Server on which you have an open session. In the Service Browser, you can:

- Select an element that you want to view.
- Select and lock an element that you want to edit.
- Copy, move, delete, or rename an element.

Elements in the Service Browser are shown in a hierarchical structure where the server is the topmost element in the hierarchy. Packages on the server contain one or more folders, which contain elements that you can create and edit using Developer (e.g., services, specifications, records).

Service Browser Icons

The icon adjacent to the name of an element denotes the element's type.
### This icon Represents...

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="server.png" alt="server_icon" /></td>
<td><strong>A server.</strong> To display the contents of the server, click the symbol next to its name. You can have multiple server contexts displayed in Developer. The active server context is the one that is highlighted in the Service Browser.</td>
</tr>
<tr>
<td><img src="package.png" alt="package_icon" /></td>
<td><strong>A package.</strong> A package contains a set of services and related files, such as specifications, records, and output templates. To display the contents of a package, click next to its name.</td>
</tr>
<tr>
<td><img src="folder.png" alt="folder_icon" /></td>
<td><strong>A folder.</strong> A folder contains related services and optional folders (called subfolders). To display the contents of a folder, click next to its name.</td>
</tr>
<tr>
<td><img src="flow_service.png" alt="flow_service_icon" /></td>
<td><strong>A flow service.</strong> A flow service is a service written in webMethods’ flow language. To display or edit the attributes of a flow service, click its name.</td>
</tr>
<tr>
<td><img src="java_service.png" alt="java_service_icon" /></td>
<td><strong>A Java service.</strong> A Java service is a service written in Java. To display or edit the Java class file for this service, click its name.</td>
</tr>
<tr>
<td><img src="webtap_service.png" alt="webtap_service_icon" /></td>
<td><strong>A WebTap service.</strong> A WebTap service provides control of an end-user’s browser experience while allowing services to intercept and modify data that passes through SAP BC Server. To display or edit the attributes of a WebTap service, click its name.</td>
</tr>
<tr>
<td><img src="c_service.png" alt="c_service_icon" /></td>
<td><strong>A C service.</strong> A C service is a service written in C/C++. To display or edit the attributes of a C service, click its name.</td>
</tr>
<tr>
<td><img src="specification.png" alt="specification_icon" /></td>
<td><strong>A specification.</strong> A specification is a formal description of a service’s inputs and outputs. To display or edit a specification, click its name.</td>
</tr>
<tr>
<td><img src="record.png" alt="record_icon" /></td>
<td><strong>A record.</strong> A record contains a set of variables. To display or edit a record, click its name.</td>
</tr>
<tr>
<td><img src="schema.png" alt="schema_icon" /></td>
<td><strong>An SAP BC schema.</strong> An SAP BC schema is the blue print or model document that you validate an XML document against. The schema defines what can and cannot be contained in the XML documents it validates. To display a schema, click its name.</td>
</tr>
</tbody>
</table>
### Refreshing the Contents of the Service Browser

The Service Browser on your screen is not dynamically updated when other users lock, unlock, add, delete, or rename elements in a package on the server. To refresh the contents of the Service Browser while you are working with Developer, click **Refresh** on the Service Browser’s toolbar.

**Refreshing the Service Browser**

![Image of the Service Browser](Image)

The Editor contains the controls that you use to examine and edit an element you select in the Service Browser. The contents of the Editor vary depending on the type of element you select.

**The Editor**

The Editor contains the controls that you use to examine and edit an element you select in the Service Browser. The contents of the Editor vary depending on the type of element you select.
If you select and lock a flow service in the Service Browser...

...the service-editing interface is displayed in the Editor.

If you select and lock a specification...

...the specification-editing interface is displayed.
Supported Data Types

SAP BC Developer supports several data types for use in services. Each data type supported by Developer corresponds to a Java data type and has an associated icon. When working in the Editor, you can determine the data type for a variable by looking at the icon next to the variable name. Two exceptions to this are the Object and Object List icons. These icons are used to represent variables whose data types are not any of the following: Strings, String Lists, String Tables, Records, or Record Lists.

Variables in a service can be any of the following data types.

<table>
<thead>
<tr>
<th>This icon...</th>
<th>Represents...</th>
<th>Java Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Icon]</td>
<td>A String. A String of characters.</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>![Icon]</td>
<td>A String List. A one-dimensional String array.</td>
<td>java.lang.String[]</td>
</tr>
<tr>
<td>![Icon]</td>
<td>A String Table. A two-dimensional String array.</td>
<td>java.lang.String[][]</td>
</tr>
<tr>
<td>![Icon]</td>
<td>A Record. A data structure that is a container for other variables. Records can contain variables of any other data type. The contents of a record are stored as key/value pairs where the variable name is the key.</td>
<td>com.wm.data.IData</td>
</tr>
<tr>
<td>![Icon]</td>
<td>com.wm.util.Values</td>
<td>For more information, see the SAP BC Java API Reference.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>A Record List. A one-dimensional array of Records (IData or Values []). Tables that are instances of com.wm.util.Table appear as Record Lists in SAP BC Developer. These tables can be used as Record Lists in flows. Services in the WmDB package use tables that are instances of com.wm.util.Table. Tables can also be declared as Objects. Objects or user-defined table-like objects that do not implement the com.wm.util.pluggable.WMIDataList interface appear as Objects in SAP BC Developer.</td>
<td>com.wm.data.IData[]</td>
</tr>
<tr>
<td>![Icon]</td>
<td>com.wm.util.Values[]</td>
<td></td>
</tr>
<tr>
<td>![Icon]</td>
<td>com.wm.util.Table</td>
<td></td>
</tr>
</tbody>
</table>
Basic Operations

Working in the Developer Window

To perform an operation on an object displayed in the Developer window, the area in which that object appears must be the “selected” area (i.e., that area must have the focus). Only one area in the Developer window can be selected at a time. Toolbar buttons and menu commands that can be used with the selected area will be available. For example, when you work in the Service Browser, it is the “selected” area—menu commands and toolbar buttons operate only within the Service Browser. Commands and buttons that cannot be used with the Service Browser are unavailable.

To switch from one area of the Developer window to another, simply click any white space or field within the area to which you want to switch. This action changes the focus to the new area and makes its menu commands and toolbar buttons available for use.

For more information about data types supported in Developer, see Appendix C, “Supported Data Types” on page 577.
Multiple Areas and Tabs

For some element types in the Service Browser, the Editor is divided into multiple areas. You switch among areas within the Editor just as you would between the Service Browser and the Editor—to select a different area, simply click any white area in that area.

Sometimes, editing controls within an area are grouped onto tabs. To display the contents of a tab, click the tab name.

Multiple Areas and Tabs in the Editor

Resizing Areas in the Developer Window

You can resize areas in the Developer window by dragging borders with your mouse. Movable borders are marked with ▲ ▼ symbols. You can also click the ▲ ▼ symbols to expand an area to the full height or width of the current window.

You can also click ☑️ to maximize the area that has focus. Click ☑️ to restore the Developer window to the normal view.
CHAPTER 2 Getting Started with Developer

Resizing the areas

Creating New Elements

To create a new element, on the File menu, click New. Follow the prompts given by Developer.

Note: You cannot create a new Java, C, or WebTap service unless all services of those types are unlocked or locked by you in the folder in which you want to create the new service. For details, see the SAP BC Cooperative Development Guide.

Locking and Unlocking Elements

To edit an element in the Service Browser, you must first lock it by right-clicking the mouse and selecting Lock. For further procedures on locking, see the SAP BC Cooperative Development Guide.

Moving and Copying Elements

You can move and copy services, specifications, and records within the Service Browser using your mouse. However, you cannot move or copy packages, folders, or WIDL services. To move an element, make sure that you have the lock or it is unlocked. Click the element in the Service Browser and drag it to the new location. You can move elements between folders and packages. To enable safeguards when moving an element, see “Renaming and Deleting Elements” on page 37.
To copy an element, select the element in the Service Browser and right-click the mouse. On the right-click menu, click Copy. Select the folder where you want to copy the element and right-click (this can be the same folder). On the right-click menu, click Paste. A copy of the element is inserted in the selected folder. If you copied the element to the same folder, “Copy” is appended to the element name. If you want to rename the element, make sure that you have the lock or it is unlocked and select it and press ENTER. You can also use the Rename command on the right-click menu.

**Important!** When you copy a Java service from one folder to another, you must re-specify the Shared tab information in the copy of the service. (You can copy the information from the Shared tab for the original service to the Shared tab for the copy of the service.) In addition, you must recompile the copy of the Java service (using jcode or the Save command in Developer) before running it on SAP BC Server.

### Renaming and Deleting Elements

If you have a lock or an element is unlocked in the Service Browser, you can rename it, with the exception of packages, folders, and WIDL services. You can also delete any element in the Service Browser.

Before you rename or delete an element, make sure that you enable the following safeguards in Developer, so that you do not inadvertently affect other elements on the SAP BC Server. This is especially important during collaborative development on the same SAP BC Server.

**Important!** You cannot use the Undo command with renaming or deleting a element.
To enable safeguards for renaming, moving, or deleting elements

1. On the Edit menu, click Preferences.
2. On the General tab, under Service Browser, do the following:

   - Select... To...
   - Confirm before deleting Instruct Developer to check for any flow services, records, or specifications that use the element.
     If elements are found that depend on the element to be deleted, then those dependents are listed. You are given the opportunity to delete the element anyway or cancel the operation.
   - Check dependency when renaming / moving Instruct Developer to check for flow services, records, and specifications that use the element.
     If elements are found that use the element to be renamed, then those dependents are listed. You are given the opportunity to:
     - Rename/move the selected element as well as all references in dependent flow services, records, and specifications.
     - Rename/move the selected element only.
     - Cancel the operation.
3. Click OK.

To rename an element

1. In the Service Browser, select the element that you want to rename. You must have the element locked or it must be unlocked.
2. Right-click and select Rename. A cursor appears in the name of the element.
3. Edit the name and click the mouse, or press ENTER.

   If you have the renaming safeguards enabled in the Preferences dialog box, Developer displays a dialog box listing all dependent elements. For example, if you renamed the PO:logPO flow service, the following dialog box may appear.
Rename Dialog Box with Dependent Elements Found

![Diagram of Rename dialog box](image)

4 Do one of the following:

- **Click...**
  - **Update**
    - Rename the selected element in the Service Browser.
  - **Update All**
    - Rename the selected element as well as all references in dependent elements. WebTap rules and WIDLs will not be renamed.
  - **Cancel**
    - Cancel the operation and preserve the original name of the element.

⚠️ **Important!** If you are renaming a schema, Developer will not find any dependents be renamed, because this functionality does not support schemas.

To delete an element

1. In the Service Browser, select the element that you want to delete. You must have the element locked or it must be unlocked.

2. Right-click and select **Delete**.

   If you enabled the deleting safeguards in the Preferences dialog box, Developer displays a dialog box listing all dependent elements. For example, if you wanted to delete the PO:logPO flow service, the following dialog box may appear.
3 Do one of the following:

<table>
<thead>
<tr>
<th>Click...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continue</td>
<td>Delete the element from the Service Browser. References in dependent elements remain.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancel the operation and preserve the element in the Service Browser.</td>
</tr>
</tbody>
</table>

**Saving Your Work**

When you open Developer and select and lock an element in the Service Browser, Developer opens a copy of that element for you to edit. Changes that you make to the element are not written to SAP BC Server until you explicitly save your work using the **Save** command.

If you attempt to close Developer or close your session on the current server without saving your changes, Developer will prompt you to save them before you exit or close your session on the server.

**Switching between Loaded Elements**

You can quickly switch between previously visited elements by choosing **View / History** (Shortcut `<CTRL>`+`<ALT>`+`'h'`) or the **History** toolbar button. This will open a dialog with a list of already visited elements:
The Toolbar Button for the History Window

![Toolbar Button for History Window](image)

Work History Window

![Work History Window](image)

The list contains up to 20 non duplicate entries. Select an entry and press Go to or double click on an entry to select the correlated service in the service tree.
**Note:** There are situations when an entry in the history dialog is not represented by a service entry in the tree view any more. This may happen when you perform some action in the Administrator UI like installing a new package version. In this case selecting the Element in the Dialog will not lead to selecting the service in the tree view. When reselecting the service in the service tree you will get a new work history entry you can use afterwards.

**Note:** The work history is not persistent. It is lost when closing the Developer.

When the checkbox *keep window open* is checked the history window stays open and you are able to switch quickly between the services you work on without needing to reopen the dialog.

**Tip!** To get a feeling for the behaviour of the work history dialog, open it and check the *keep window open* checkbox. Now click on some services in the service tree. You can see the services to be added to the history.

**Reverting Changes**

If you made a change in an element, but want to cancel that change before saving it, use the *File / Revert* function. You will also find the *Revert* action in the popup menu of any service in the service tree.

After starting the *Revert* action and pressing *OK* on the following dialog, the service is reloaded from the server in its former state.
Handling Deprecated Elements

Viewing a Deprecated Service

In the Settings tab of an element and also in the icon for the tree item you get the information if an element is deprecated. This means the element should not be used anymore, because it may be removed in a future release.
A Deprecated Service used in a Flow Service

When using a deprecated service in your own flow service, you see a deprecated service marked with a special icon in the flow view. The deprecation note can be found in the Properties panel of the Service.

In the Flow Diagram View you find the deprecation icon in the rectangle representing the flow step and the comment for the deprecated service in the fly-over text of the rectangle.
The Deprecation Mark in the Flow Diagram View

In your own packages, you can also set this deprecation information in the **Settings** tab / Deprecation on delivery.

**Setting deprecation on delivery** flag and comment
CHAPTER 2 Getting Started with Developer

After publishing the package using the Publishing/Create and Delete Releases link in the Administrator, this information will be shown in the Deprecation section of the Settings tab.

Note: You can deprecate any service using this mechanism. Deprecating a record definition, a schema, a specification or a template is not possible.

Note: The deprecation feature only works if you are connected to a SAP Business Connector Server 4.8 or higher.

Service after Publishing and Reinstalling the Package

[Image of the Service Output Template screen showing the use of deprecated service and its settings.]
Opening, Closing, and Restoring Sessions

When you start Developer you are prompted to log on to the server that you want to work with. You maintain a session on that server until you exit Developer or close the session.

You can have open sessions on multiple servers at a time. The selected server in focus in the Service Browser is the current server on which your commands will be executed. For example, if you have the localhost:5555 server selected in the Service Browser and you select the New command, the new element will be created on that server.

You can open a session on another server without closing your current session by using the Open Session command.

⚠️ Important! While you have an open session on a server through Developer, you are using a licensed seat for that server. At times when you are not actively using Developer, you may want to close your session on the server to free a seat on the server for others to use.

To close a session on the current server
1. Save any work that you want to keep.
2. On the File menu, select Close Session.
   –OR–
   Click ✗ on the Service Browser toolbar.

To open a session on a different server
1. On the File menu, select Open Session.
   –OR–
   Click ☑ on the Service Browser toolbar.
2. Complete the Open Session dialog box. If you need procedures for this step, see “To start Developer” on page 27.
3. Click OK.
Restoring a Session on a Server

Sometimes a server might shut down before you can save your work. Developer preserves any unsaved work as well as lock information, despite the loss of the connection to the server. When the server restarts, you can restore your session and save your changes to the server.

⚠️ Important! If you close your session (disconnect from the current server) or exit Developer before the server restarts, all unsaved work will be lost.

To restore a session on the server

- On the File menu, select Restore Session.

Notification of Server Shutdown

If the server administrator shuts down the server on which you have an open session, Developer does one of the following:

- If the server administrator specified a time delay before shutdown, Developer displays a message notifying you about the number of minutes until shutdown occurs. (This message also includes the time when the shut down was started.) After you receive notification of server shutdown, save any work that you want to keep and then close your session. If you do not close your session, Developer notifies you when the server has shut down.

- If the server administrator performed an immediate shutdown, Developer displays a message stating that your connection to the server has been lost. (Developer also displays this message if the network connection to the server is lost.)

If you did not save your work before shut down occurred, you might be able to restore your session when the server restarts and then save your work. For more information about restoring sessions, see the previous section.

Changing Your Password

You can change the password for your user account. If you forget your password, contact the server administrator.

⚠️ Important! Do not change your password if you are outside of the corporate firewall and you did not use SSL to open the session on the SAP BC Server.
Password Requirements

For security purposes, SAP BC Server places length and character restrictions on passwords. SAP BC Server contains a default set of password requirements, however, your server administrator can change these. Contact your server administrator for information about password requirements for SAP BC Server.

The default password requirements provided by SAP are as follows:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum Length</td>
<td>8</td>
</tr>
<tr>
<td>Minimum Number of Alphabetic Characters</td>
<td>3</td>
</tr>
<tr>
<td>Minimum Number or Uppercase Characters</td>
<td>2</td>
</tr>
<tr>
<td>Minimum Number of Lowercase Characters</td>
<td>2</td>
</tr>
<tr>
<td>Minimum Number of Numeric Characters</td>
<td>1</td>
</tr>
<tr>
<td>Minimum number of special characters (non-alphabetic and non-numeric characters, such as *, ?, &amp;)</td>
<td>1</td>
</tr>
</tbody>
</table>

To ensure the security of your password, follow the additional guidelines below:

- Do not choose obvious passwords, such as your name, address, phone number, license plate, spouse's name, child's name, or a birthday.
- Do not use any word that can be found in the dictionary.
- Do not write your password down.
- Do not share your password with anyone.
- Change your password frequently.

Examples of good passwords include: RuKIDDing?, 4theLUVo$, two4TEA?.

To change your password

1. On the File menu, select Change Password.
2. In the Change Password dialog box, in the Old Password field, type your current password.
3. In the New Password field, type your new password.
4 In the **Confirm New Password** field, retype your new password. Click **OK**.

---

**Important!** The Server Administrator can disable the feature for changing your password from Developer. If the feature is disabled, when you try to change your password, you will receive a message stating that the administrator has disabled the feature.

---

### Caching Elements

You can improve performance in Developer by caching Service Browser elements that are frequently used. You can also clear the cache of elements at any time. To do this, you use the **Caching** tab on the **Preferences** dialog box.

---

#### To cache elements

1 On the **Edit** menu, click **Preferences**.
2 Click the **Caching** tab.

#### Partial Caching Tab on Preferences Dialog Box

![Partial Caching Tab on Preferences Dialog Box](image)

3 In the **Number of elements to cache** field, type the number of elements that you want to cache per session. The minimum number of elements is 10. The higher the number of elements, the more likely an element will be in the cache, which reduces network traffic and speeds up Developer.

4 Click **OK**. The caching settings take effect immediately for all servers on which you have open sessions.

---

**Important!** The Server Administrator can disable the feature for changing your password from Developer. If the feature is disabled, when you try to change your password, you will receive a message stating that the administrator has disabled the feature.

---

#### Clearing the Developer Cache

When you clear the Developer cache, you remove Service Browser elements from memory for all servers on which you have open sessions. The following elements are not removed:

---

**Note:** Keep in mind that increasing the cache reduces memory. If you experience memory problems, consider reducing the number of elements that are cached.
Flow services with breakpoints (to fix, remove the breakpoint and clear the cache again)

Flow services that are currently being debugged (for example, if a service has been stepped into)

Unsaved elements

Keep in mind that the cache is automatically cleared when you close Developer or when you refresh the session by clicking the refresh button on the toolbar.

To clear the Developer cache

1. On the Edit menu, click Preferences. The Preferences dialog box appears.
2. Click the Caching tab if it is not already in the foreground.
3. Click the Clear Cache button. All cached elements are removed from memory.

Note: Clearing cached elements from Developer is different from clearing the contents of the pipeline from SAP BC Server. If you want to clear the Server cache, see the Settings tab in the Editor area. For more information, see “Configuring a Service’s Use of Cache” on page 88.

Using Online Help

You can access online Help at any point in SAP BC Developer by clicking the Help button (available in most dialog boxes) or pressing F1. You can also use the On Topic and Contents commands on the Help menu.
CHAPTER 3

Working with Packages

- What Is a Package? ................................................................. 54
- Packages Installed with SAP BC Server ................................. 54
- Package Management .......................................................... 55
- Assigning Startup, Shutdown, and Replication Services ........... 65
CHAPTER 3 Working with Packages

What Is a Package?

A package is a container that is used to bundle services and related elements, such as specifications, records, SAP BC schemas, and output templates. When you create a folder, service, specification, record, SAP BC schema, or output template, you save it in a package.

Packages are designed to hold all of the components of a logical unit in an integration solution. For example, you might group all the services and files specific to a particular marketplace in a single package. By grouping these components into a single package, you can easily manipulate them as a unit. For example, you can copy, reload, distribute, or delete the set of components—the package—with a single operation.

Although you can group services using any package structure that suits your purpose, most sites organize their packages by function or application. For example, they might put all purchasing-related services in a package called “PurchaseOrderMgt” and all time-reporting services into a package called “TimeCards.”

On the server, a package represents a subdirectory within the <sapbc>\server\packages directory. All the components that belong to a package reside in the package’s subdirectory.

Note: Every service in SAP BC Developer must belong to a package.

Packages Installed with SAP BC Server

SAP BC Server comes with the following predefined packages:

- **Default.** This package is empty. SAP provides it as a convenience so you can create services and store them here without first creating a package.

- **SAP.** This package contains the services that you need for administration of the SAP BC Server. Also, you’ll find all the services regarding RFC, BAPI, IDoc and XML communication in this package.

- **WmDB.** This package contains services that you can use to access JDBC-enabled databases. Do not alter or delete this package.

- **WmPartners.** This package contains services for Partner Manager. For information about the services in this package, see the SAP BC SAP Adapter Guide, which is located in the <sapbc>\server\packages\WmPartners\pub\doc directory on SAP BC Server.

- **WmPublic.** This package contains services that you can safely invoke from your client applications and services. For more information about the services in this package, see the SAP BC Built-In Services Guide.

- **WmSamples.** This package contains sample services. The tutorial folder contains services used in the SAP BC Developer Tutorial.
WmWin32. This package contains services you can use to invoke methods on COM objects. This package also contains Windows-specific samples, such as sample Visual Basic services. The WmWin32 package is installed, but is not enabled when you install SAP BC Server. For information about enabling a package, see the SAP BC Administration Guide.

Note: The names of predefined packages that contain services, records, specifications, and other elements begin with the prefix "Wm".

Package Management

SAP BC Developer provides several tasks that you can perform to manage the packages on SAP BC Server. When you perform a package management task, it affects all of the files and services in the package. You can complete basic package management tasks, such as creating and deleting a package, using Developer. Only the server administrator can perform certain complex tasks, such as enabling, disabling, archiving, publishing, and recovering a package.

The following table identifies all of the package management tasks that can be performed using Developer or the Server Administrator. If you can perform the task with Developer, the See column directs you to a page with the instructions. If a person using the Server Administrator can perform the task, the See column directs you to the SAP BC Administration Guide.

<table>
<thead>
<tr>
<th>To...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a package</td>
<td>page 57</td>
</tr>
<tr>
<td>Activate a package</td>
<td>SAP BC Administration Guide</td>
</tr>
<tr>
<td>Lock the contents of a package</td>
<td>SAP BC Cooperative Development Guide</td>
</tr>
<tr>
<td>Reload the services and files in a package into memory without restarting the server</td>
<td>page 59 and SAP BC Administration Guide</td>
</tr>
<tr>
<td>Enable a package that you previously disabled</td>
<td>SAP BC Administration Guide</td>
</tr>
<tr>
<td>Disable access to a package, but do not want to delete the package</td>
<td>SAP BC Administration Guide</td>
</tr>
<tr>
<td>Delete all services and related files in a package</td>
<td>page 59 and SAP BC Administration Guide</td>
</tr>
<tr>
<td>Recover the services and related files from a deleted package.</td>
<td>SAP BC Administration Guide</td>
</tr>
</tbody>
</table>
## To...  
Assign a version number to a package  
Identify packages that must be loaded before a specific package is loaded (package dependencies)  
Export a package or partial package  
Replicate or copy the contents of a package and send (publish) it to other SAP BC Servers  
Archive a copy of the package (such as for a backup copy)  
View the patch history for a package  
Assign startup, shutdown, or replication services to a package

<table>
<thead>
<tr>
<th>To...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign a version number to a package</td>
<td>page 61 and SAP BC Administration Guide</td>
</tr>
<tr>
<td>Identify packages that must be loaded before a specific package is loaded (package dependencies)</td>
<td>page 63</td>
</tr>
<tr>
<td>Export a package or partial package</td>
<td>page 60</td>
</tr>
<tr>
<td>Replicate or copy the contents of a package and send (publish) it to other SAP BC Servers</td>
<td>SAP BC Administration Guide</td>
</tr>
<tr>
<td>Archive a copy of the package (such as for a backup copy)</td>
<td>SAP BC Administration Guide</td>
</tr>
<tr>
<td>View the patch history for a package</td>
<td>page 62 and SAP BC Administration Guide</td>
</tr>
<tr>
<td>Assign startup, shutdown, or replication services to a package</td>
<td>page 67</td>
</tr>
</tbody>
</table>
Creating a Package

When you want to create a new grouping for services and related files, create a package. Packages can store services, specifications, records, output templates, and schemas.

Important! To create a package, you must have Developer or Administrator privileges.

When you create a package, Developer creates a new subdirectory for the package in the file system on the machine where SAP BC Server is installed. For information about the subdirectory and its contents, see the SAP BC Administration Guide.

Keep the following guidelines in mind when naming new packages:

- Start all package names with an uppercase letter and capitalize the first letter of subsequent words. For example, PurchaseOrder.
- Keep package names short. Use abbreviations instead of full names. For example, instead of ProcessPurchaseOrder, use ProcessPO.
- Make sure the package name describes the functionality and purpose of the services it contains.
- Avoid creating package names with random capitalization. For example, cOOLPkgTest.
- Avoid using articles (e.g., “a,” “an,” and “the”) in the package name. For example, instead of TestTheService, use TestService.
- Avoid using the prefix “Wm”. Developer uses the “Wm” prefix for predefined packages that contain services, records, and other files.

To create a package

1. Click the Service Browser toolbar.
   -OR-
   On the File menu, click New.
2. In the New dialog box, select Package, and click Next.
3. In the New Package dialog box, in the Name field, type the name for the new package using any combination of letters, numbers, and the underscore character. Click Finish. Developer automatically refreshes the Service Browser and displays the new package.
CHAPTER 3 Working with Packages

Documenting a Package

You can communicate the purpose and function of a package and its services to other developers by documenting the package. To make the package documentation accessible to other developers, use Web documents (such as HTML pages) for the documentation and place the documents in the pub subdirectory for the package on SAP BC Server.

For example, place the package documentation for a package named “PurchaseOrders” in the following directory: <sapbc>\server\packages\PurchaseOrders\pub

Tip! An alternate location for package documentation is the <sapbc>\server\packages\doc directory. Typically, this directory is used for reference material such as PDFs that do not need to be published to the Web.

For each package, make sure to create an index.html file for use as a home page for the package documentation. The index.html file can contain links to the other Web documents for the package. An index.html file exists for each package installed by the SAP BC Server.

Use the following process to access the documentation for a package.

To access documentation for a package

- Enter the URL for the package documentation. The URLs for package documentation have the following format:

  http://serverName:port\PackageName\DocumentName

  where:

  serverName:port is the name and port address of the SAP BC Server on which the package resides.

  PackageName is the name of the package for which you want documentation.

  DocumentName is the name of the Web document you want to access. If you do not specify a DocumentName, SAP BC Server automatically displays the index.html file.

Note: Avoid using control characters and special characters like periods (.) in the name of a package. The watt.server.illegalNSChars setting in the server.cnf file (which is located in the <sapbc>\server\config directory) defines all the characters that you cannot use when naming packages. Additionally, the operating system on which you run SAP BC Server might have specific requirements that limit package names.
**Reloading a Package**

Sometimes, you need to reload a package on the server to activate changes that have been made to it outside of Developer. You need to reload a package if any of the following occurs:

- A Java service that was compiled using jcode is added to the package
- New jar files are added to the package
- Any of the configuration files for the package are modified

**Note:** Reloading a package is not the same thing as refreshing the Service Browser. When you refresh the Service Browser, SAP BC Developer retrieves a fresh copy of the contents of all the packages from the memory of SAP BC Server. When you reload a package, SAP BC Server removes the existing package information from memory and loads new versions of the package and its contents into its memory.

### To reload a package

- In the Service Browser, right-click the package you want to reload, and click **Reload Package**.
  
  --OR--

- In the Service Browser, select the package you want to reload, and click **File > Reload Package**.

**Deleting a Package**

When you no longer need the services and files in a package, you can delete the package. Deleting a package removes the package and all of its contents from the Service Browser.

When you delete a package from Developer, SAP BC Server saves a copy of the package. If you later want to recover the package and its contents, contact your server administrator. Only Server Administrator users can recover a package. For more information about recovering packages, see the *SAP BC Administration Guide*. 
Before you delete a package, make sure of the following:

- Other users or other services do not use (depend on) the services, templates, records, and schemas in the package. You can use the Find Dependents to identify other services that are dependent on a service in a package that you want to delete. For more information, see “Finding Dependents” on page 240.

- All elements are unlocked in the package that you want to delete. If there are elements that are locked by others or system locked, you cannot delete the package.

**To delete a package**

- In the Service Browser, right-click the package you want to delete, and click Delete.

  -OR–

- In the Service Browser, select the package you want to delete, and click Edit ➤ Delete.

**Exporting a Package or Partial Package**

Packages or parts of a package, such as a folder, can be exported to your hard drive so that they can be shared with partners or developers. You can install an exported package on another server by using the package publishing functionality in the Server Administrator. Locking information is not exported.

**To export a package**

1. In the Service Browser, select the package you want to export and select File ➤ Export.

2. In the Export To dialog box, select the location on your hard drive where you want the exported package to reside. Click Save.

   This exports the package to a ZIP file and saves it on your hard drive. The ZIP file can then be published on another server.

**To export a partial package**

1. In the Service Browser, select the folder or element that you want to export, and select File ➤ Export.

2. In the Export To dialog box, select the location on your hard drive where you want the exported partial package to reside. Click Save.

   This exports the folder or element to a ZIP file and saves it on your hard drive. The ZIP file can then be published on another server.
Assigning a Version Number for a Package

You can assign a version number to a package to identify different versions of the package. For example, you might want to assign a new version number to a package when you add new services to a package or after you fix bugs in a package. You might find assigning version numbers especially helpful if you work in a development environment where more than one person makes changes to a package.

By default, Developer assigns the version number 1.0 to each package that you create.

**Important!** When you change the version number of a package, make sure that you update the package dependencies for other packages that depend on the earlier version of this package.

**Tip!** Assign and change package version numbers through Developer only when the packages are in a development stage. To avoid difficulties installing package releases, do not change version numbers on packages you receive from trading partners, packages to which you subscribe, or packages installed with SAP BC Server.

To assign a version number to a package

1. In the Service Browser, select the package to which you want to assign a version number.
2. Click the Settings tab.
3. In the Package Version field, type the version number you want to assign to the package. The version number needs to adhere to one of the following patterns: `x.x` or `x.x.x` (e.g., `1.0`, `2.1`, `2.1.3`, or `3.1.2`)
4. Click to save your changes.

   If the version number you entered does not adhere to one of the formats specified in step 3, Developer displays a message stating that the format is not correct.

**Note:** You can also use the Server Administrator to assign version numbers to packages. For more information, see the *SAP BC Administration Guide.*
CHAPTER 3  Working with Packages

Viewing the Patch History for a Package

For each package, Developer tracks and displays the history of installed patches. A patch is partial upgrade, change, or fix to the contents of a package.

You might want to check a package’s patch history for the following reasons:

- To avoid overwriting the installed package with a lower version of the same package.
- To view the changes that comprise each version of the package.
- To inform SAP Customer Care which versions of predefined packages are installed on your SAP BC Server.

When you select a package in the Service Browser, the **Settings** tab displays the patch history since the last full release of the package. (A full release of a package incorporates all previous patches for the package.)

The **Settings** tab displays patch history for the package

Note: With the exception of the **Package version** field and the fields under **Package dependencies**, the fields on the **Settings** tab are display-only.

Note: When the server administrator installs a full release of a package (a release that includes all previous patches for the package), SAP BC Server removes the existing patch history. This helps the server administrator avoid potential confusion about version numbers and re-establish a baseline for package version numbers.
To view patch history for a package

- In the Server Administrator, select the package for which you want to view a patch history. Click the **Settings** tab.

The following table describes each field under **Patch history**:

<table>
<thead>
<tr>
<th>This field...</th>
<th>Specifies...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>The name of the package.</td>
</tr>
<tr>
<td><strong>Version</strong></td>
<td>The version number of the package. A user assigns a version number when they create a package release. By default, Developer assigns version 1.0 to a new package.</td>
</tr>
<tr>
<td><strong>Build</strong></td>
<td>The build number of the package. The build number is a generation number that a user assigns to a package each time the package is regenerated. For example, a user might generate version 1.0 of the “Finance” package ten times and assign build numbers 1,2,3...10 to the different generations or builds of the package. The Build number is not the same as the Version number. One version of a package might have multiple builds.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>A brief description of the package written by the user who created the package release.</td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td>The time at which the package release (patch) was created.</td>
</tr>
<tr>
<td><strong>JVM Number</strong></td>
<td>The version of the JVM (Java virtual machine) required to run the package.</td>
</tr>
<tr>
<td><strong>Publisher</strong></td>
<td>The name of the publishing server that created the package release.</td>
</tr>
<tr>
<td><strong>Patch Number</strong></td>
<td>The patch numbers included in this release of the package.</td>
</tr>
</tbody>
</table>

**Identifying Package Dependencies**

If a package needs the services in another package to load before it can load, you need to set up package dependencies. For example, suppose that you have a package named “Finance.” The “Finance” package contains a service that invokes a service in the “FinanceUtil” package. For the service in “Finance” to execute, the “FinanceUtil” package needs to be loaded. To ensure that the “FinanceUtil” package loads before the “Finance” package, you need to identify the “FinanceUtil” package as a package dependency for the “Finance” package.
Tip! You might also want to identify package dependencies if a startup service for a package invokes a service in another package. The startup service cannot execute if the package containing the invoked service has not yet loaded.

To identify package dependencies for a package

1. In the Service Browser, select the package for which you want to specify package dependencies.
2. Click the Settings tab.
3. Under Package Dependencies, click .
4. In the Enter Input Values dialog box, enter the following information:

<table>
<thead>
<tr>
<th>In this field...</th>
<th>Specify...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>The name of the package you want SAP BC Server to load before the package selected in the Service Browser.</td>
</tr>
<tr>
<td>Version</td>
<td>The version number of the package you want loaded. To identify multiple versions of the same package, use an asterisk (<em>) in the version number. For example, to specify all versions of a package with a version number that starts with “3.”, type 3.</em> for the version number. To specify all versions with a version number that starts with “3.1”, type 3.1.* for the version number.</td>
</tr>
</tbody>
</table>

5. Click OK and then click on the toolbar.

Important! Make sure that you do not create circular package dependencies. For example, if you identify “FinanceUtil” as a dependent package for the “Finance” package, do not identify “Finance” as a dependent package for the “FinanceUtil” package. If you create circular package dependencies, neither package will load the next time you start SAP BC Server.

Removing Package Dependencies

Use the following procedure to remove a package dependency that is no longer needed. For example, to continue the example from page 63, if you delete the service in “Finance” that invokes the service in “FinanceUtil,” then you would no longer need a package dependency on the “FinanceUtil” package. You might also remove the package dependency if you move the services in the “FinanceUtil” package into the “Finance” package.
To remove a package dependency

1  In the Service Browser, select the package for which you want to remove a package dependency.

2  Click the Settings tab.

3  Under Package Dependencies, select the package dependency you want to remove.

4  Click .

Assigning Startup, Shutdown, and Replication Services

You can set up services to automatically execute each time SAP BC Server loads, unloads, or replicates a package. These types of services are called startup, shutdown, or replication services.

What Is a Startup Service?

A startup service is one that SAP BC Server automatically executes when it loads a package into memory. The server loads a package:

- At server initialization (if the package is enabled).
- When someone uses Developer or the Server Administrator to reload a package.
- When someone uses Developer or the Server Administrator to enable a package.

Startup services are useful for generating initialization files or assessing and preparing (e.g., setting up or cleaning up) the environment before the server loads a package. However, you can use a startup service for any purpose. For example, you might want to execute a time-consuming service at startup so that its cached result is immediately available to client applications.

Tip! If a startup service invokes a service in another package, make sure to identify the other package as a package dependency for the package containing the startup service.
What Is a Shutdown Service?
A shutdown service is one that SAP BC Server automatically executes when it unloads a package from memory. The server unloads a package from memory:

- At server shutdown or restart.
- When someone uses the Server Administrator to disable the package.
- When someone uses the Server Administrator to reload a package before it is removed from memory.

Shutdown services are useful for executing clean-up tasks such as closing files and purging temporary data. You could also use them to capture work-in-progress or state information before a package unloads.

What Is a Replication Service?
A replication service is one that SAP BC Server automatically executes when it prepares to replicate a package. A replication service executes when the Server Administrator creates a package release (full release or patch) or creates a package archive.

Replication services provide a way for a package to persist state or configuration information so that these are available when the published package is activated on the remote server.

Note: The term replication service does not refer to the services contained in pub.replicator or to services that subscribe to replication events (replication event services).

Guidelines for Assigning Startup, Shutdown, and Replication Services
Keep the following guidelines in mind when assigning startup, shutdown, and replication services to packages:

- When you assign a startup or shutdown service to a package, you can only assign a service that resides in the same package. For example, a startup service for the “Finance” package must be located in the “Finance” package.
- When you assign a replication service to a package, you can assign any service from any loaded package on SAP BC Server, including the current package.
Because services in a package are not made available to clients until the package’s startup services finish executing, you should avoid implementing startup services that access busy remote servers. They will delay the availability of other services in that package.

You can assign one or more startup services to a package; however, you cannot specify the order in which the services execute. If you have a series of startup services that need to execute in a specific order, create a “wrapper” service that invokes all the startup services in the correct order. Designate the “wrapper” service as the startup service for the package.

Identifying Startup, Shutdown, and Replication Services

Use the following procedure to identify startup, shutdown, and replication services.

To identify startup, shutdown, and replication services

1. In the Service Browser, select the package to which you want to assign startup, shutdown, or replication services.
2. Click the Startup/Shutdown/Replication Services tab.
3. To add a startup service, under Startup services, select the service from the Available Services list, and click . Repeat this step for each service you want to add as a startup service for the package.

Note: A service that you just created does not appear in the Available Services list if you have not refreshed your session on the server since you created the service.

4. To add a shutdown service, under Shutdown services, select the service from the Available Services list, and click . Repeat this step for each service you want to add as a shutdown service for the package.
5. To add a replication service, do the following:
   1. Under Replication Services click .
   2. In the Enter Input Values dialog box, in the service field, do one of the following:
      - Type the service name in the format: folderName:serviceName
      - Click to navigate to and select the service you want to use as a replication service.
   3. Click OK.
   4. Repeat steps 1–3 for each service you want to add as a replication service.
Removing Startup, Shutdown, and Replication Services

Use the following procedure to remove startup, shutdown, and replication services. You might need to remove a startup, shutdown, or replication service if the service is no longer needed, has been deleted, or has been incorporated into another service (such as a wrapper service).

Tip! If you remove a startup service that invoked a service in another package and the package was identified as a package dependency, make sure you remove the package dependence after you remove the startup service.

To remove startup, shutdown, and replication services

1. In the Service Browser, select the package for which you want to remove startup, shutdown, or replication services.

2. Click the Startup/Shutdown/Replication Services tab.

3. Do one or more of the following:
   - To remove a startup service, under Startup services, select the service you want to remove from Selected services list, and click [ ].
   - To remove a shutdown service, under Shutdown services, select the service you want to remove from the Selected services list, and click [ ].
   - To remove a replication service, under Replication services, select the replication service you want to remove and click [ ].
Building Flow Services

- Basic Concepts .................................................. 70
- A Process Overview ............................................ 75
- Creating a New Flow Service ................................. 76
- Declaring Input and Output Parameters for a Service ........................................... 81
- Specifying Run-Time Parameters .............................. 87
- Assigning an Output Template to a Service ................... 91
- Managing Security for Services ................................. 93
- Assigning Universal Names to Services ....................... 98
- Printing a Flow Service .......................................... 99
Basic Concepts

To successfully build a flow service, you should understand the following basic concepts and terms.

What Is a Flow Service?

A flow service is a service that is written in webMethods’ flow language. This simple yet powerful language lets you encapsulate a sequence of services within a single service and manage the flow of data among them. For example, you might create a flow service that takes a purchase order from a buyer and executes the following series of services before submitting it to an internal ordering system:

1. Gets a purchase order submitted by a buyer
2. Logs the order in an audit-trail file
3. Performs a credit check
4. Posts the order to the ordering system

Flow services encapsulate other services

Any service can be invoked within a flow (including other flow services). For instance, a flow might invoke a service that you create, any of the built-in services provided by webMethods, and/or services from a webMethods add-on product such as the webMethods SAP R/3 Adapter or the webMethods Baan Adapter.

You create flow services using Developer. They are saved in XML files on SAP BC Server.

Important! Although flow services are written as XML files, they are maintained in a format that can only be created and understood by Developer. You cannot create or edit a flow service with a text editor.
Flow Services and WIDL

In SAP BC Developer 3.x, flow services replaced WIDL services as the standard framework for building business applications. If you have WIDL services, they will continue to run on SAP BC Server 4.x. However, if you need to edit a WIDL service, you must do so using SAP BC Developer 2.x.

Once you install SAP BC Developer 4.x, we strongly recommend that you use it to create all new services as flow services. Flow services provide the same capabilities as WIDL services, plus much more.

What Is a Flow Step?

A flow service contains flow steps. A flow step is a basic unit of work (expressed in webMethods flow language) that SAP BC Server interprets and executes at run time. webMethods’s flow language provides flow steps that invoke services and flow steps that let you edit data in the pipeline.

webMethods’ flow language also provides a set of flow-control steps that allow you to direct the execution of a flow service at run time. The flow-control steps allow you to:

- Conditionally execute a specified sequence based on a variable value.
- Retry a specified sequence until it succeeds.
- Repeat a specified sequence (loop) for each element in an array variable.

In the following flow service, flow-control steps have been inserted to loop through a subset of the flow service and branch to one of two services in the last step of the loop.
Flow-control steps are used to direct the execution of a flow.

A LOOP step repeats a set of flow steps.

A BRANCH step selects a specified flow step for execution.

A flow service can contain the following types of flow steps:

**Invocation Steps**

- **INVOKE**
  
  Executes a specified service. For more information about this step, see “Invoking Services” on page 107.

**Data-Handling Steps**

- **MAP**
  
  Performs specified editing operations on the pipeline (e.g., mapping variables in the pipeline, adding variables to the pipeline, dropping variables from the pipeline, and so forth). For more information about this step, see “The MAP Step” on page 132.

**Flow-Control Steps**

- **BRANCH**
  
  Executes a specified flow step based on the value of a specified variable in the pipeline. For more information about this step, see “The BRANCH Step” on page 110.

- **LOOP**
  
  Executes a set of flow steps once for each element in a specified array. For more information about this step, see “The LOOP Step” on page 126.

- **REPEAT**
  
  Re-executes a set of flow steps up to a specified number of times based on the successful or non-successful completion of the set. For more information about this step, see “The REPEAT Step” on page 118.
SEQUENCE

Groups a set of flow steps into a series. The SEQUENCE step is implicit in most flow services (that is, the steps in a flow service are treated as a series). However, at times it is necessary to explicitly group a subset of flow steps using SEQUENCE so that they can be treated as a unit. For more information about this flow step, see “The SEQUENCE Step” on page 125.

EXIT

Controls the execution of a flow step; for example, abort an entire flow service from within a series of deeply nested steps, throw an exception without writing a Java service, or exit a LOOP or REPEAT without throwing an exception. For more information about this step, see “The EXIT Step” on page 130.

See Appendix A, “SAP BC Flow Steps” for a detailed description of each type of flow step provided by the webMethods flow language. For information about building each type of flow step, see “Creating an SAP Inbound Map” on page 77.

What Is the Pipeline?

The pipeline is the general term used to refer to the data structure in which input and output values are maintained for a flow service. It allows services in the flow to share data.

The pipeline starts with the input to the flow service and collects inputs and outputs from subsequent services in the flow. When a service in the flow executes, it has access to all data in the pipeline at that point.
The pipeline holds the input and output for a flow service

When you build a flow service, you use Developer to specify how information in the pipeline is mapped to and from services in the flow.

**What Are Input and Output Parameters?**

Input and output parameters are the names and types of variables that the service requires as input and generates as output. For example, a service that takes two string values—an account number (AcctNum) and a dollar amount (OrderTotal)—as input and produces an authorization code (AuthCode) as output, has the following input and output parameters:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Data Type</strong></td>
</tr>
<tr>
<td>AcctNum</td>
<td>String</td>
</tr>
<tr>
<td>OrderTotal</td>
<td>String</td>
</tr>
<tr>
<td>AuthCode</td>
<td>String</td>
</tr>
</tbody>
</table>

Part of the process of creating a service is declaring its input and output parameters—i.e., explicitly specifying the variables it expects as input and produces as output.
A Process Overview

Building a flow service is a process that involves the following basic stages:

- **Stage 1**: Creating a new service on SAP BC Server. During this stage, you create the new service on the SAP BC Server where you will do your development and testing. For information about this stage, see “Creating a New Flow Service” which follows.

- **Stage 2**: Inserting flow steps into the new service. During this stage, you specify the work that you want the service to perform by adding flow steps to the service. For information about this stage, see “Creating an SAP Inbound Map” on page 77.

- **Stage 3**: Declaring the input and output parameters of the service. During this stage, you define the service's inputs and outputs. For information about this stage, see “Declaring Input and Output Parameters for a Service” on page 81.

- **Stage 4**: Mapping pipeline data. During this stage, you route input and output variables between services that are invoked in the flow. For information about this stage, see “Mapping Data in a Flow Service” on page 149.

- **Stage 5**: Specifying the run-time parameters. During this stage, you assign parameters that configure the run-time environment for this service. For information about this stage, see “Specifying Run-Time Parameters” on page 87.

- **Stage 6**: Formatting service output. During this stage you can create an output template to format the service output. For information about this stage, see “Assigning an Output Template to a Service” on page 91 or refer to the Building Output Templates and DSPs.

- **Stage 7**: Testing and debugging. During this stage you can use the tools provided by Developer to test and debug your flow service. For information about this stage, see “Testing and Debugging Services” on page 247.

The remaining sections in this chapter and the following chapters provide detailed information about each stage.
Creating a New Flow Service

The first step you take when you build a flow service is to create a new service on the SAP BC Server where you do your development.

Package and Folder Requirements

Before you create a new flow service, you must:

- Make sure the package in which you want to create the flow service already exists. If the package does not already exist, create it using Developer. For more information about creating a package, “Creating a Package” on page 57.

- Make sure the folder in which you want to create the service already exists. If the folder does not already exist, create it using Developer. For step-by-step procedures, see online Help.

Once the package and folder are in place, you use the File > New command to start the wizard that leads you through the process of creating a new service. For step-by-step procedures, see online Help on flow services.

Using the Default Logic Options

The wizard that creates a new flow service gives you the option of starting with an empty flow (one that contains no predefined flow steps) or starting with a service that contains a set of default logic for extracting information from an HTML or XML document (a common flow-service function).

If you are building a flow service that extracts data from an HTML or XML document, you can select an option to automatically generate the logic (i.e., the set of flow steps) that will get data from a specified document, rather than building this logic manually.

The wizard also allows you to create a flow service that contains an SAP Outbound Map. In this case the flow service will execute a function module in an SAP system.
Creating an SAP Inbound Map

If you want an SAP system to execute a service on the SAP BC you can create an SAP Inbound Map. An SAP Inbound map associates an incoming function call with an SAP BC service that is to be executed. The procedure to create an SAP Inbound Map is slightly different from an SAP Outbound Map (see “Using the Default Logic Options” on page 76), as in case of an Inbound Map you do not choose an arbitrary name.

To create an SAP Inbound Map
1. In the Service Browser, select the New button to create a new element.
2. Select SAP Inbound Map in the New window and click Next.
3. Select a package for the Inbound Map and click Next.
4. Enter the Inbound Map generation parameters in the window New SAP Inbound Map and click Finish.

Generating Elements via SAP Lookup

With SAP BC 4.7 you can generate elements for the developer via SAP lookup functions. This allows you to retrieve information on function modules, structures (tables), BAPIs and IDocs from an SAP System and directly generate outbound maps or records for them.

SAP Lookup functions

If you use one of these Lookup functions you can display the structure of the corresponding SAP data elements. SAP BC Developer uses the following icons to display these structures:

This Icon... means:

- IDoc Type
- IDoc Segment
<table>
<thead>
<tr>
<th>This Icon...</th>
<th>means:</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Icon" /></td>
<td>Data Type CHAR</td>
</tr>
<tr>
<td><img src="image2" alt="Icon" /></td>
<td>Data Type NUMBER</td>
</tr>
<tr>
<td><img src="image3" alt="Icon" /></td>
<td>Data Type DATE</td>
</tr>
<tr>
<td><img src="image4" alt="Icon" /></td>
<td>Data Type TIME</td>
</tr>
<tr>
<td><img src="image5" alt="Icon" /></td>
<td>Data Type RAW</td>
</tr>
<tr>
<td><img src="image6" alt="Icon" /></td>
<td>Data Type STRUCTURE</td>
</tr>
<tr>
<td><img src="image7" alt="Icon" /></td>
<td>Business Object Repository</td>
</tr>
<tr>
<td><img src="image8" alt="Icon" /></td>
<td>Business Object</td>
</tr>
<tr>
<td><img src="image9" alt="Icon" /></td>
<td>BAPI (BOR)</td>
</tr>
<tr>
<td><img src="image10" alt="Icon" /></td>
<td>Keyfield (BOR)</td>
</tr>
<tr>
<td><img src="image11" alt="Icon" /></td>
<td>Import Parameter</td>
</tr>
<tr>
<td><img src="image12" alt="Icon" /></td>
<td>Export Parameter</td>
</tr>
<tr>
<td><img src="image13" alt="Icon" /></td>
<td>Changing Parameter</td>
</tr>
<tr>
<td><img src="image14" alt="Icon" /></td>
<td>Function Module</td>
</tr>
<tr>
<td><img src="image15" alt="Icon" /></td>
<td>Table Parameter</td>
</tr>
<tr>
<td><img src="image16" alt="Icon" /></td>
<td>Data Type Table</td>
</tr>
</tbody>
</table>
To lookup a Function Interface and generate an Outbound Map

1. Choose SAP -> Function Interface from the menu.
2. Select an SAP Server and the Function Module you are looking for. Click on Lookup.
3. The structure of the function module is displayed now.
4. Click on Generate Outbound Map if you want to save the function module as an outbound map in SAP BC Developer.

To lookup a Structure Definition and generate a Record

1. Choose SAP -> Structure Definition from the menu
2. The structure is displayed now.
3. Click on Generate Record if you want to save the structure as a record.

To lookup a Business Object

1. Choose SAP -> BOR Browser from the menu
2. Select the SAP Server you want to retrieve the BOR information from and click on Lookup.
3. Select the Business Object(s) you are looking for.

To lookup an IDoc and generate a record

1. Choose SAP -> IDoc Definition from the menu.
2. Select an SAP Server and enter the IDoc type you are looking for. Click on Lookup.
3. The IDoc structure is displayed now. If you want to generate a record from it, click on Generate record.

To browse the Function Maps specified in the SAP BC Developer
1. Choose SAP -> Function Maps from the menu.
2. This will browse all function maps (inbound and outbound) that are specified in the SAP BC Developer.

**Inserting Flow Steps**

Flow steps call previously built services and direct the flow of data within a flow service. You can insert flow steps into a flow service using the buttons at the top of the Flow Editor.

**Click this button...**  **To insert...**

- An INVOKE step.
- A MAP step.
- A BRANCH step.
- A LOOP step.
- A REPEAT step.
- A SEQUENCE step.
- An EXIT step.

For more information about inserting flow steps, see “Building Flow Services” on page 69.

**Declaring Input and Output Parameters for a Service**

Input and output parameters are the names and data types of the variables that a service expects as input and generates as output. Some systems refer to input and output parameters as “imports” and “exports.” These parameters are also collectively referred to as a signature.

You declare input and output parameters for all types of services: flow services, Java services, and services written in other supported programming languages.
Although you are not required to declare input and output parameters for a service (SAP BC Server will execute a service regardless of whether it has a specification or not), there are good reasons to do so:

- **Declaring parameters makes the service’s input and outputs visible to Developer.** Without declared input and output parameters, you cannot:
  - Map data to and/or from the service using Developer's Pipeline Editor.
  - Assign default input values to the service with the Pipeline Editor.
  - Validate the input and output values of the service at run time.
  - Test the service in Developer and enter initial input values.
  - Generate skeleton code for invoking the service from a client.

- **Declaring parameters makes the input and output requirements of your service known to other developers** who may want to call your service from their programs.

For these reasons, we strongly recommend that you make it a practice to declare input and output parameters for every service that you create.

### Specifying Input Parameters

When you define the input parameters for a flow service, keep the following points in mind:

- **Specify all inputs that a calling program must supply to this flow service.** For example, if a flow service invokes two other services, one that takes a variable called `AcctNum` and another that takes `OrderNum`, you must define both `AcctNum` and `OrderNum` as input parameters for the flow service.

**Note:** The purpose of declaring input parameters is to define the inputs that a calling program or client must provide when it invokes this flow service. You do not need to declare inputs that are obtained from within the flow itself. For example, if the input for one service in the flow is derived from the output of another service in the flow, you do not need to declare that variable as an input parameter.

- **When possible, use variable names that match the names used by the services in the flow.** Variables with the same name are automatically mapped to one another in the pipeline. (Remember that variable names are case-sensitive.) If you use the same variable names used by flow’s constituent services, you reduce the amount of manual data mapping that needs to be done. When you specify names that do not match the ones used by the constituent services, you must use the Pipeline Editor to manually map them to one another. For information about the Pipeline Editor, see “What Is the Pipeline Editor?” on page 150.

- **Make sure the variables match the data types of the variables they represent in the flow.** For example, if a service in the flow takes a Record List called `LineItems`, define that input
variable as a Record List. For a complete description of the data types supported by a service, see Appendix C, “Supported Data Types” on page 577.

- **Declared input variables appear automatically as inputs in the pipeline.** When you select the first service or MAP step in the flow, the declared inputs appear under **Pipeline In**.

**Important!** If you edit a cached service by changing the inputs (not the pipeline), you must click **Reset Cache** on the **Settings** tab in Developer, or in **Service Usage** in the Server Administrator. If you do not reset the Server cache, the old cached input parameters will be used at run time.
CHAPTER 4  Building Flow Services

Specifying Output Parameters

On the output side of the Input/Output tab you specify the variables that you want the flow to return to the calling program or client. The guidelines for defining the output parameters are similar to those for defining input parameters:

- **Specify all of the output variables that you want this flow service to return** to the calling program or client.

- **Make sure the names of output variables match the names used by the services** that produce them. Like input variables, if you do not specify names that match the ones produced by the flow’s constituent services, you must use the Pipeline Editor to manually map them to one another. For information about the Pipeline Editor, see “What Is the Pipeline Editor?” on page 150.

- **Make sure the variables match the data types of the variables they represent in the flow.** For example, if a service produces a Record List called LineItems, make sure you define that variable as a Record List or a Record. For a complete description of the data types supported by a service, see Appendix C, “Supported Data Types” on page 577.

- **Declared output variables appear automatically as outputs in the pipeline.** When you select the last service or MAP step in a flow, the declared output variables appear under Pipeline Out.

Completing the Input/Output Tab

You declare the input and output parameters for a service using the Input/Output tab. On the left side of this tab, you define the variables that the service requires as input. On the right side, you define the variables the service returns to the client or calling program.
Declaring Input and Output Parameters for a Service

Input/Output tab

For a flow service, the input side describes the initial contents of the pipeline. In other words, it specifies the variables that this flow service expects to find in the pipeline at run time. The output side identifies the variables produced by the flow service and returned to the pipeline.

You can complete the Input/Output tab in the following ways:

- **Reference a specification.** A specification defines a set of service inputs and outputs. You can use a specification to define input and output parameters for multiple services. When you assign a specification to a service, you cannot add, delete, or modify the declared variables using the service’s Input/Output tab.

- **Reference a record.** You can use a record to define the input or output parameters for a service. When you assign a record to the Input or Output side of the Input/Output tab, you cannot add, modify, or delete the variables on that half of the tab.

- **Manually insert input and output variables.** Use □□□ to specify the data type and name for each input and output variable for the service.
To declare input and output parameters for a service

1. In the Service Browser, select the service for which you want to declare input and output parameters.

2. In the Editor, click the Input/Output tab.

3. If you want to reference a specification, do the following:

   1. In the Specification Reference field, type the specification's name or click ![ ] to select it from a list. For more information about creating specifications, see “Declaring Input and Output Parameters for a Service” on page 81.

   2. If you assigned a record to both the Input and Output sides of the Input/Output tab, skip the rest of this procedure. Otherwise, continue to the next step to specify the variables in the other half of the tab.

Important! When a specification is assigned to a service, you cannot add, delete, or modify the declared variables using the service's Input/Output tab.

4. If you want to reference a record for the input or output parameters of the service, do the following:

   1. In the Input or Output field (depending on which half of the specification you want to assign the record to), type the record name or click ![ ] to select it from a list.

   2. If you assigned a record to both the Input and Output sides of the Input/Output tab, skip the rest of this procedure. Otherwise, continue to the next step to specify the variables in the other half of the tab.

Important! When a record is assigned to the Input or Output side, you cannot add, delete, or modify the declared variables on that half of the Input/Output tab.

5. For each input or output variable that you want to define, do the following:

   1. Select the half of the Input/Output tab (Input or Output) where you want to define the variable by clicking anywhere in that half's large white text box.

   2. Click ![ ] on the toolbar and select the type of variable that you want to define.

   3. Type the name of the variable and press ENTER.

   4. Right-click the variable and select Properties to set variable properties and apply validation constraints (optional).

   5. If the variable is a Record or a Record List, repeat steps 2–4 to define each of its members. Use ![ ] to indent each member beneath the Record or Record List variable.
6 If you want to enter any notes or comments about the input and output parameters, type your comments in the **Comments** field.

## Specifying Run-Time Parameters

As a developer of a service, you can use the options on the **Settings** tab to specify whether or not you want the server to treat it as a “stateless” service at run time and/or to cache its results.

You can also cache elements to reduce memory usage in Developer. For details, see “To cache elements” on page 50.

---

**Important!** The run-time options on the **Settings** tab should only be set by someone who is thoroughly familiar with the structure and operation of the selected service. Improper use of these options can lead to a service failure at run time and/or the return of invalid data to the client program.

---

## Maintaining the State of a Service

When a remote client opens a session on a SAP BC Server, the server automatically builds a session object for that client. The server uses this object to maintain specific information about the client requesting the service. It contains information such as user name and password. The server maintains the session object for the duration of the session—i.e., until the client program explicitly closes the session on the server or times out due to inactivity.

When you develop services in a language such as Java, C/C++, or Visual Basic, you can use the “put” method to write information to the session object. You might do this to store information that a sequence of services needs to maintain a connection to an external system.

A service that is an atomic unit of work—i.e., one that is wholly self contained and not part of a multi-service transaction to an external system—does not to need to have its session object maintained when it is finished executing. For performance reasons, you may want to configure these types of services to run in “stateless” mode. Services that run in “stateless” mode use fewer resources and do not consume a licensed seat on SAP BC Server.

---

**Important!** Do not use the stateless option unless you are certain that the service operates as an atomic unit of work. If you are unsure, leave the **Stateless** option disabled.
To configure a service's run-time state

1. In the Service Browser, select the service that you want to configure.
2. Select the Settings tab.
3. Under Runtime, select the Stateless check box if you do not want the server to maintain session information for this service. Otherwise, leave this option disabled.

Configuring a Service’s Use of Cache

Caching is an optimization feature that can improve the performance of stateless services. When you enable caching for a service, SAP BC Server saves the entire contents of the pipeline after invoking the service in a local cache for the period of time that you specify. When the server receives subsequent requests for a service with the same set of input values, it returns the cached result to the client rather than invoking the service again. Caching can significantly improve response time of services. For example, services that retrieve information from busy data sources such as high-traffic commercial Web servers could benefit from caching. The server can cache the results for all types of services—flows, Java services, and C/C++ services.

Note: Caching is only available for data that can be written to the repository server. Since nodes cannot be written to the repository, they cannot be cached.

Types of Services to Cache

While caching service results can improve performance, not all services should be cached. You should never cache services if the cached results might be incorrect for subsequent invocations or if the service performs tasks that must be executed each time the service is invoked. This section describes guidelines for you to consider when determining whether to cache the results for a service.

Services Suited for Caching

- **Services that require no state information.** If a service does not depend on state information from an earlier transaction in the client’s session, you can cache its results.
- **Services that retrieve data from data sources that are updated infrequently.** Services whose sources are updated on a daily, weekly, or monthly basis are good candidates for caching.
- **Services that are invoked frequently with the same set of inputs.** If a service is frequently invoked by clients using the same input values, it is beneficial to cache the results.
Services That You Should Not Cache

- **Services that perform required processing.** Some services contain processing that must be processed each time a client invokes it. For example, if a service contains accounting logic to perform charge back and you cache the service results, the server does not execute the service, so the service does not perform charge back for the subsequent invocations of the service.

- **Services that require state information.** Do not cache services that require state information from an earlier transaction, particularly information that identifies the client that invoked it. For example, you do not want to cache a service that produced a price list for office equipment if the prices in the list vary depending on the client who initially connects to the data source.

- **Services that retrieve information from frequently updated sources.** If a service retrieves data from a data source that is updated frequently, the cached results can become outdated. Do not cache services that retrieve information from sources that are updated in real-time or near real-time, such as stock quote systems or transactional databases.

- **Services that are invoked with unique inputs.** If a service handles a large number of unique inputs and very few repeated requests, you will gain little by caching its results. You might even degrade server performance by quickly consuming large amounts of memory.

Controlling a Service’s Use of Cache

You use the options on the **Settings** tab to enable caching and to configure the way in which you want it to operate with the selected service. You use these settings to strike the right balance between data currency and memory usage. To gauge the effectiveness of your cache settings, you can monitor its performance by viewing service statistics with the Server Administrator and then adjusting your caching values accordingly.

**Note:** If you do not have administrator privileges on your SAP BC Server, work with your server administrator to monitor and evaluate your service’s use of cache.

**Important!** If you edit a cached service by changing the inputs (not the pipeline), you must click **Reset Cache** on the **Settings** tab in Developer, or in **Service Usage** in the Server Administrator. If you do not reset the server cache, the old cached input parameters will be used at run-time.
CHAPTER 4 Building Flow Services

Specifying the Duration of a Cached Result
The server maintains results in cache for the period of time you specify in the Cache Expire field on the Settings tab. The expiration timer begins when the server initially caches a result, and it expires when the time you specify elapses. (The server does not reset the expiration timer each time it satisfies a service request with a cached result.) The minimum cache expiration time is one minute.

Note: The cache may not be refreshed at the exact time specified in Cache Expire. It may vary from 0 to 15 seconds, according to the cache sweeper thread. For details, see the watt.server.cache.flushMins setting in SAP BC Server.

Using the Prefetch Option
You use the Prefetch option to specify whether or not you want the server to automatically refresh the cache for this service when it expires. If you enable Prefetch, the server automatically re-executes the service (using the same set of inputs as before) to update its results in cache. This action also resets the cache expiration timer.

Important! Use Prefetch carefully. Overuse can quickly exhaust the memory available for cache.

Important! Do not use Prefetch with Java or C/C++ services that invoke access-controlled services. Such services will fail during prefetch because the embedded service will be invoked without the proper access privileges. To avoid this problem, enable Prefetch on the invoked services rather than on the Java or C/C++ services that call them.

When you enable Prefetch, you must also set the Prefetch Activation option to specify when the server should initiate a prefetch. This setting specifies the minimum number of times a cached result must be accessed (hit) in order for the server to prefetch results. If the cache receives less than the number of hits you specify, the server will not perform a prefetch when the results expire.

The following procedure describes how to use Developer to enable caching for a service.

Note: The cache may not be refreshed at the exact time the last hit fulfills the Prefetch Activation requirement. It may vary from 0 to 15 seconds, according to the cache sweeper thread. For details, see the watt.server.cache.flushMins setting in SAP BC Server.

To enable caching of pipeline contents after a service is invoked

1. In the Service Browser, select the service that you want to configure.
2. Select the Settings tab.
3 Under **Runtime**, select the **Caching** check box.

4 In the **Cache Expire (minutes)** field, type an integer representing the length of time (in minutes) that you want the results for this service to be available in cache.

5 If you want to use prefetch, do the following:
   1. Select the **Prefetch** check box.
   2. In the **Prefetch Activation (hits)** field, specify the minimum number of hits needed to activate the use of prefetch.

### Assigning an Output Template to a Service

An output template is a Web document, such as an HTML page, embedded with special codes (tags) that SAP BC Server processes. These tags instruct SAP BC Server to perform a specific action and substitute the result of that action in the document. Typically, you use tags in output templates to insert service output values in documents returned to clients.

Output templates are used most frequently to customize the HTML page that a service returns to a browser-based application. However, they can also be used to generate an XML document or any other formatted string. For example, you may have a service that retrieves a record from a relational database and uses an output template to format it as an XML document or a comma-delimited record before returning it to the requestor.

**Note:** If a service has an output template assigned to it, the server automatically applies the template to the results of the service (i.e., the contents of the pipeline) whenever that service is invoked by an HTTP client. If a service does not have an output template, the server simply returns the results of the service in the body of an HTML document, formatted as a two-column table.

When using output templates, keep in mind that:

- You do not have to assign an output template to a service.
- A service can have at most one output template assigned to it at a time (you can dynamically change the output template assignment at run time, however).
- You can assign the same output template to more than one service.
- If you assign an existing output template to a service, the output template must reside in the `<sapbc>\server\packages\packageName\templates` directory where `packageName` is the package in which the service is located.
- You can reference one output template from within another.
To assign an output template to a service

1. In the Service Browser, select the service to which you want to assign an output template.

2. Click the Settings tab.

3. In the Name field, do one of the following:
   - If you want to assign a new output template to the service, type the name of the new output template. By default, Developer assigns the service an output template with the name FolderName_ServiceName.
   - If you want to assign an existing output template to the service, type the file name of the existing output template. You do not need to include the path information or the file name extension.

4. In the Type list, do one of the following to specify the format for the output template:

   Select... To...
   html       Assign an HTML output template to the service.
   xml        Assign an XML output template to the service.
   wml        Assign a WML output template to the service.
   hdml       Assign an HDML output template to the service.

   Note: The Type you select determines the extension for the output template file (*.html, *.xml, *.wml, or *.hdml). In cases where the output is returned to a Web browser, the Type also determines the value of the HTTP “Content-Type” header field (e.g., “text/html,” “text/xml,” “text/vnd.wap.wml,” or “text/x-hdml”).

5. Do one of the following:
   - If you assigned a new output template to the service, click New to create the output template.
   - If you assigned an existing output template to the service and you want to edit the template, click Edit.
Managing Security for Services

As a developer of services, you can control access to the services that clients can invoke by assigning Access Control Lists (ACLs) to services. An ACL identifies the groups that are allowed to access a service. When a client requests that SAP BC Server invoke a service, the server checks the ACL assigned to the service. If the client is a member of an allowed group and not a member of a denied group, the server executes the service. If the client is not a member of an allowed group, the server denies the request to invoke the service and stops executing.

By default, when a client requests a service, SAP BC Server only checks the ACL of the externally invoked service (the service requested directly by the client). The server does not check the ACLs of any of the internally invoked services—those services invoked by the externally invoked service. However, you can set up the security settings for a service so that SAP BC Server checks the ACL assigned to the service every time it is invoked, whether directly by a client or by another service.

The following diagram illustrates the points at which ACL checking occurs when a client requests a service.

ACL checking when a client requests a service
### Assigning ACLs to Folders and Services

You can assign ACLs to folders, subfolders, and services. When you assign an ACL to a folder, it affects the subfolders and services in the folder. The subfolders and services that do not have an assigned ACL inherit the ACL that you assign to the folder. (Subfolders and services with an assigned ACL are not affected by the ACL assigned to the folder.) When a subfolder or service inherits the ACL of a folder, the **Access Control List (ACL)** field displays an asterisk (*) after the assigned ACL name.
When you create root folders (the top-level folders in a package), Developer automatically assigns the Internal ACL to the folder. Services protected by the Internal ACL are accessible only to users belonging to the Administrators or Developers groups. If you are developing a service that will be externally invoked by users (such as your partners), you may want to assign an ACL other than Internal to the service or folder that contains the service.

The following sections explain how to assign ACLs to folders and services. For guidelines and recommendations about setting up security for services, see the SAP BC Administration Guide.

Assigning an ACL to a Folder

You can assign an ACL to any folder in the Service Browser. Remember, the subfolders and services in a folder inherit the ACL you assign to the folder. Make sure the ACL you assign to the folder sufficiently protects as many services in the folder as possible. If the folder contains a service that you think merits stronger security, assign a more restrictive ACL to the individual service.

To assign an ACL to a folder

1. In the Service Browser, select the folder or subfolder to which you want to assign an ACL.
2. On the Settings tab, select the ACL you want to assign to the folder or subfolder from the Access Control List (ACL).
3. Click .

For information about the default ACLs and guidelines for creating and selecting an ACL, see the SAP BC Administration Guide.

Assigning an ACL to a Service

You can specifically assign an ACL to a service. If you do not assign an ACL to a service, the service inherits the ACL assigned to the folder in which the service is contained.

When you assign security settings to a service, you can use the Enforce ACL on Internal Invokes option to require that SAP BC Server check the ACL of the service every time the service is invoked—even when the service is internally invoked. (SAP BC Server always checks the ACL of the service when a client directly invokes the service.)
To assign an ACL to a service

1. In the Service Browser, select the service to which you want to assign an ACL.
2. Click the Settings tab.
3. Under Security, in the Access Control List (ACL), select the ACL you want to assign to the service.
4. Next to Enforce ACL on Internal Invokes, select one of the following options:

   - **Select...**
     - **On**: Instruct SAP BC Server to always check the ACL of the service against the permissions of the group(s) to which the requesting client belongs.
     - If you select this option, the server checks the ACL of the service every time it is invoked—even when the service is invoked by another service.
     - **Off**: Instruct SAP BC Server to ignore the ACL of the service when it is invoked internally (invoked by another service).

     This is the recommended option.

5. Click .

**Requiring ACL Checking for Internal Invokes**

The following procedure explains how to enforce ACL checks on internal invocations of the service.

To require an ACL check on internal invokes of a service

1. In the Service Browser, select the service for which you want to require an ACL check for internal invocations.
2. Click the Settings tab.
3. Next to Enforce ACL on Internal Invokes, select the **On** option. This instructs SAP BC Server to perform a ACL check when the service is invoked by another service.
4. Click .
Removing ACLs from a Folder or Service

Use the following procedure to remove an ACL from a folder or service. When you remove an ACL from a service or subfolder, the service or subfolder inherits the ACL assigned to the folder in which the service or subfolder is located. When you remove the ACL assigned to the root folder (the uppermost folder in a package), SAP BC Server applies the Default ACL to the folder and its contents for which an ACL is not specified. (The Default ACL restricts access to a service to any user with a valid username and password for SAP BC Server.)

To remove an ACL from a folder or service

1. In the Service Browser, select the folder or service from which you want to remove an ACL.
2. Click the Settings tab.
3. In the Access Control List (ACL), select <None>.
4. Click .

The As-User Property in SAP BC Developer 3.x

In SAP BC Developer 3.x, you could specify an as-user property when building a flow service. At run time, this property allowed a client to invoke a child service in a flow using access rights different than the client’s access rights.

Because of security enhancements, the as-user property is no longer necessary. Consequently, when processing client requests for services created in version 3.x, SAP BC Server version 4.0 ignores the as-user property for any INVOKE steps.
Assigning Universal Names to Services

Every service on a SAP BC Server has a universal name in addition to its regular SAP BC name. A universal name is a unique public identifier that external protocols use to reference a service on a SAP BC Server. A universal name has two parts: a namespace name and a local name.

- The namespace name is a qualifier that distinguishes a SAP BC service from other resources on the Internet with the same local name. For example, there could be many resources with the name AcctInfo. A namespace name distinguishes one AcctInfo resource from another by specifying the name of the collection to which it belongs (similar to the way in which a state or province name serves to distinguish cities with the same name—for example, Springfield, Illinois, versus Springfield, Ontario).

Like namespaces in XML, the namespace portion of a universal name is usually specified as a URI. This convention assures uniqueness, because URIs are based on globally unique domain names. However, it does not have to be a URI; it can be composed of any sequence of letters or digits. The following, for example, are all valid namespace names:

- http://www.gsx.com
- myNamespaceName
- gl.journals.cashTransactions

- The local name uniquely identifies a service within a particular namespace. Most sites use the service’s unqualified name as its local name. Under this scheme, for example, a service named gl.journals.closeGL would have a local name of closeGL. The Integration Server does not require you to use the service name as a local name; however, you can specify the local name as any sequence of letters, or digits. For example, all of the following would be valid local names for a service called orders:postOrder:

  - postOrder
  - PO
  - orders.add.regularPO

For details on universal names, see the SAP BC SOAP Programming Guide.

To assign, edit, or view a universal name

1. In the Service Browser, select the service whose universal name you want to assign, edit, or view.

2. Click the Settings tab.
If you want to assign or edit the service's universal name, specify the following under Universal Name:

**In this field...** | **Specify...**
--- | ---
**Namespace name** | The name that will be used to qualify the name of this service. You may use any sequence of characters or digits for the namespace name.  
- **Note:** By convention, a URI is generally used as the namespace name (e.g., http://www.gsx.com/gl). This assures that the universal name is globally unique.

**Local name** | A name that uniquely identifies the service within the collection encompassed by **Namespace name**. You may use any sequence of characters or digits for the local name.  
- **Note:** Most sites use the unqualified portion of the service name as the local name.

Click ![save icon](save.png) to save the new settings.

To delete a universal name:

1. In the Service Browser, select the service whose universal name you want to delete.
2. Click the **Settings** tab.
3. Under **Universal Name**, remove the current settings from the **Namespace name** and **Local name** fields.
4. Click ![save icon](save.png) to save the new settings.

### Printing a Flow Service

The following procedure describes how to use the **View as HTML** command to produce a printable version of a flow service. This lets you see all aspects of a flow—its input and output parameters, its flow steps, and pipeline behavior—in a single document.
A flow report lets you view all aspects of the flow service at once

PoMDW_SGxOrderRec

Service Input
- node
  - TransactionID
  - SourceServer

Service Output
- none

Flow Overview
1. InVOICE documentToRecord
2. InVOICE debugLoco
3. MAP
4. InVOICE getFile
5. InVOICE getAuthCode
6. MAP
7. InVOICE PostRO_SGx

Flow Detail
1. InVOICE xmlWeb:documentToRecord
   Convert MDW XML PO to Sales Order

<table>
<thead>
<tr>
<th>Service Out</th>
<th>Pipeline Out</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>boundNode</td>
</tr>
<tr>
<td></td>
<td>P0_In</td>
</tr>
</tbody>
</table>

To print a flow service

1. In the Service Browser, select the service that you want to print.
2. From the File menu, select View as HTML.
3. If you want to print the flow, select your browser’s print command.

Note: When you print a flow service as HTML while viewing the flow in diagram view, only the flow diagram appears. Service input, service output and flow details do not appear. Additionally, only the flow steps currently visible in the Flow Editor appear in the resulting HTML page. To fit all of the steps in a flow service into the Flow Editor (and therefore, into a single HTML page), click to zoom out of the flow service or click to maximize the Flow Editor.
Inserting Flow Steps

- Inserting and Moving Flow Steps .......................................................... 102
- Invoking Services ............................................................................... 107
- The BRANCH Step ............................................................................. 110
- The REPEAT Step .............................................................................. 118
- The SEQUENCE Step ......................................................................... 125
- The LOOP Step .................................................................................. 126
- The EXIT Step ................................................................................... 130
- The MAP Step ................................................................................... 132
A flow step is the basic unit of work that instructs SAP BC Server about what to do with data at each stage of a flow service. Flow steps can invoke services and direct the course of execution. Using flow steps you can:

- Invoke a service, such as a flow service, Java service, C service, or Web Service Connector (INVOKE).
- Conditionally execute one step from a set of specified alternatives (BRANCH).
- Repeat a set of flow steps up to a specified number of times or until a step in the set fails or succeeds as specified (REPEAT).
- Group a set of flow steps and control the way in which the failure of a member of the set is processed (SEQUENCE).
- Repeat a set of flow steps over the elements of a specified array (LOOP).
- Exit the entire flow or exit a single flow step (EXIT).
- Map, add, edit, and delete pipeline variables or invoke several services that operate on the same set of pipeline variables (MAP).

**Inserting and Moving Flow Steps**

To insert flow steps in a flow service, you must first select that service in the Service Browser and then click the **Flow** tab. The flow steps in the service are listed in the “Flow Editor” at the top of the tab. (If you just created the service, and it does not contain any default logic, the Flow Editor is empty).
Inserting and Moving Flow Steps

Flow Steps for the selected flow service are displayed in the Flow Editor

Note: When creating flow services, you can use the Flow Tree view or the Flow Diagram view of the Flow Editor. (Flow Tree view is shown in the above graphic. In Flow Diagram view, a flow service looks like a flow chart.) Flow Tree view is the default view for the Flow Editor. Consequently, the procedures in this book are written for working in Flow Tree view. For information about working in Flow Diagram view, see “Using Flow Diagram View” on page 133.

Flow steps are inserted into a service using the following toolbar buttons at the top of the Flow Editor (you cannot directly type a flow step into the Flow Editor—all editing is performed through the toolbar):

<table>
<thead>
<tr>
<th>Click this button...</th>
<th>To insert...</th>
<th>For more information, see page...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An INVOKE step.</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td>A MAP step.</td>
<td>132</td>
</tr>
<tr>
<td></td>
<td>A BRANCH step.</td>
<td>110</td>
</tr>
</tbody>
</table>
CHAPTER 5 Inserting Flow Steps

<table>
<thead>
<tr>
<th>Click this button...</th>
<th>To insert...</th>
<th>For more information, see page...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A LOOP step.</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td>A REPEAT step.</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>A SEQUENCE step.</td>
<td>125</td>
</tr>
<tr>
<td></td>
<td>An EXIT step.</td>
<td>130</td>
</tr>
</tbody>
</table>

**Note:** If you select an existing step in the Flow Editor before inserting a new one, Developer inserts the new step below the one that is selected. If you do not have a step selected, it adds the new step to the end of the flow. You can move a step to a new location using the arrow buttons on the toolbar. See “Changing the Position of a Flow Step” which follows.

### Changing the Position of a Flow Step

Flow steps run in the order in which they appear in the Flow Editor. To move a step up or down in a flow service, you select that step in the Flow Editor. Then use the arrow buttons on the toolbar to move the step up or down in the list.

<table>
<thead>
<tr>
<th>Click this button...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Move the flow step up in the list.</td>
</tr>
<tr>
<td></td>
<td>Move the flow step down in the list.</td>
</tr>
</tbody>
</table>

You can also move a flow step by dragging it up or down with your mouse or by using the **Shift** commands on the **Compose** menu.

### Changing the Level of a Flow Step

Some flow steps have subordinate steps on which they operate. Subordinate steps are referred to as *children*. For example, when you use the LOOP step, the set of steps that make up the loop are referred to as children of that LOOP step.

Children are specified by indenting them beneath their parent flow step. In the following example, a LOOP step has three children. Note that one of its children is a BRANCH step, which has its own set of children.
Child steps are indented beneath their parent step

To promote or demote a flow step within a parent/child hierarchy, select the step in the Flow Editor, and then use one of the following buttons to move it left or right beneath the current parent step.

Click this button... To...

Demote a flow step in the hierarchy. This action makes the selected step a child of the preceding parent step. (This button will only be available if you select a step that can become a child.)

Promote a flow step in the hierarchy. This action moves the step one level up in the hierarchy

Setting the Properties of a Flow Step

Every flow step has a unique set of properties associated with it. The properties for a flow step are displayed on the Properties tab. Values that you specify on the Properties tab are applied only to the selected step in the Flow Editor.
CHAPTER 5 Inserting Flow Steps

Properties of a Flow Step

Although each type of flow step has a set of unique properties, they all have the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comment</td>
<td>Assigns an optional descriptive comment to the selected flow step.</td>
</tr>
<tr>
<td>label</td>
<td>Assigns a name to the selected flow step. When a label is assigned, that label appears next to the step in the Flow Editor. The label allows you to reference that flow step in other flow steps. In addition, you use the label to control the behavior of certain flow steps. For example, the BRANCH step uses the label property to determine which alternative it is supposed to execute. See “The BRANCH Step” on page 110 and “The EXIT Step” on page 130 for additional information about this use of the label property.</td>
</tr>
</tbody>
</table>

For a complete description of the properties associated with each flow step, see Appendix A, “SAP BC Flow Steps”.
Invoking Services

You use the INVOKE step to request a service within a flow. You can use the INVOKE step to:

- Invoke any type of service, including other flow services and Web Service Connectors.
- Invoke any service for which the caller of the current flow has access rights on the local SAP BC Server.
- Invoke built-in services and services on other SAP BC Servers.
- Invoke flow services recursively (i.e., a flow service can call itself). If you use a flow service recursively, bear in mind that you must provide a means to end the recursion.
- Invoke any service, validating its input and/or output. For details, see “Performing Input/Output Validation” on page 226.

Flow service containing four INVOKE steps

Specifying the Service Property

The INVOKE step’s service property specifies which service will be invoked at run time. When you insert an INVOKE step, Developer automatically assigns the name of that service to the service property.

If you want to change the service assigned to an INVOKE step, you edit the service property. When you specify the service property, keep the following points in mind:

- You must specify the service’s fully qualified name in folderName:serviceName format.
  
  Example: purchasing.orders:getOrders

- You must specify the service’s name exactly as it is defined on the server. Service names are case-sensitive.
CHAPTER 5 Inserting Flow Steps

Invoking a Built-In Service

There is an extensive set of built-in services that you can invoke from a flow service. The SAP BC library includes services for doing such things as transforming data values, performing simple mathematical operations, extracting information from Web documents, and accessing databases.

Built-in services reside in the WmPublic package. For a complete description of these services, see the SAP BC Built-In Services Guide.

Note: If you are using one of webMethods Adapters (e.g., webMethods BAAN Adapter, webMethods Siebel Adapter), you will have additional built-in services provided. See the documentation provided with those packages for details.

Invoking a Service on Another SAP BC Server

You can use the built-in service pub.remote:invoke to invoke a service on a remote SAP BC Server and return the results. The remote server is identified by an alias, which is configured on the Remote Servers tab in the Server Administrator. The pub.remote:invoke service automatically handles opening a session and authentication on the remote server.

The pub.remote:invoke service resides in the WmPublic package and requires the alias of the remote server and the fully qualified name of the service that you want to invoke as input. For a complete description of this service, see the SAP BC Built-In Services Guide.

Building an INVOLVE Step

Use the following procedure to invoke a service within a flow service.

To insert an INVOLVE step

1. In the Service Browser, select the flow service in which you want to invoke another service. In the Flow Editor, select the step immediately above which you want to insert the INVOLVE step.

2. Click on the Flow Editor toolbar. Select the service you want to invoke. If the service you want to invoke does not appear in the list, select Browse to navigate to and select the service.
3 Complete the following fields on the Properties tab.

**For this property...** Specify...

**service**

The service that will be invoked at run time. When you insert a service, Developer automatically assigns the name of that service to the service property. If you want to change the service that is invoked, specify the service’s fully qualified name in the `folderName:serviceName` format.

**validate-in**

Whether or not you want to validate the input to the service. Select $default to validate the input to the service. Select $none if you do not want to validate the input to the service.

For information about validating input, see “Performing Input/Output Validation” on page 226.

**validate-out**

Whether or not you want to validate the output of the service. Select $default to validate the output of the service. Select $none if you do not want to validate the output of the service.

For information about validating output, see “Performing Input/Output Validation” on page 226.

4 If necessary, on the Pipeline tab, map Pipeline In variables to Service In variables. Map Service Out variables to Pipeline Out variables. For more information about mapping variables to a service, see “Mapping Variables” on page 155.

---

**Note:** When you install Developer, the **Insert** menu displays a list of commonly-used services. You can use the **Edit > Preferences** command to customize this list of services to suit your needs.

**Note:** In SAP BC version 3.x, you could set an as-user property for a transformer. In SAP BC Server version 4.0, this property was removed. For more information see “The As-User Property in SAP BC Developer 3.x” on page 97.
Finding an Invoked Service

You can navigate to the location of an invoked service in both the flow tree view and the flow diagram view. This is especially helpful when working with a flow written by another party and with complex flows that make multiple invokes.

To find an invoked service

1. Select the invoked service you want to locate.
2. Right-click the highlighted service, and select Go To. Developer locates and selects the service in the Service Browser.

The BRANCH Step

The BRANCH step allows you to conditionally execute a step based on the value of a variable at run time. For example, you might use a BRANCH step to process a purchase order one way if the PaymentType value is “CREDIT CARD” and another way if it is “CORP ACCT”.

When you build a BRANCH step, you can:

- **Branch on a switch value.** Use a variable to determine which child step executes. At run time, the BRANCH step matches the value of the switch variable to the label property of each of its targets. It executes the child step whose label matches the value of the switch.

- **Branch on an expression.** Use an expression to determine which child step executes. At run time, the BRANCH step evaluates the expression in the label property of each child step. It executes the first child step whose expression evaluates to “true.”

Branching on a Switch Value

When you branch on a switch value, you branch on the value of a single variable in the pipeline. To build a BRANCH step that branches on a switch value, create a list of the conditional steps (target steps) and make them children of the BRANCH step. Then, complete the following:

- In the **Properties** tab for the BRANCH step, specify the name of the pipeline variable whose value will act as the switch.

- In the **label** property of each target step, specify the value that will trigger the step.
The BRANCH Step

Simple BRANCH Step that branches on a switch value

Specifying the Switch Variable

The variable you use as the switch variable:

- Must be a string variable.
- Must be a variable that exists in the pipeline when the BRANCH step is executed at run time.
- Must be formatted as record/recordElement, if you are specifying a record element as the switch variable (e.g., SupplierInfo/AccountNum).

Specifying the Label Value

At run time, the BRANCH step matches the value of the switch variable to the label property of each of its targets. It executes the step whose label matches the value of the switch.

You may use a regular expression to specify the matching value for a BRANCH step. To do this, use the following syntax to specify the value in label:

/RegularExpression/
For example, if you want to select a step based on whether a PO number starts with the string “REL” you use /\^REL/ as the value of label.

Unlike other flow steps, whose children execute in sequence at run time, only one child of a BRANCH step is executed—the target whose label matches the value of the switch variable. If none of the targets match the switch variable, none of them are performed, and execution “falls through” to the next step in the flow service. For example, in the following flow, execution passes directly to the Orders:LogTransaction service if the value of PaymentType is “COD.”

An unmatched value will fall though the BRANCH

Keep the following points in mind when assigning labels to the targets of the BRANCH step:

- You must give each target step a label unless you want to match an empty string. For that case, you leave the label property blank. For more about matching an empty string, see “Branching on Null and Empty Values” on page 115.

- Each label value must be unique within the BRANCH step.

- When you use specify a literal value as the label of a child step, the value you specify must match the run-time value of the switch variable exactly. The label property is case-sensitive.

- You may use a regular expression as the value of label instead of a literal value.

- You can use the $null option in the label property to match a null value.

- You can use the $default value in the label property to designate a default step for all unmatched cases. For more information about using the $default setting, “Specifying a Default Step” on page 114.
Branching on an Expression

When you branch on an expression, you assign an expression to each child of a branch step. At run time, the BRANCH step evaluates the expressions assigned to the child steps. It executes the first child step with an expression that evaluates to true.

To build a BRANCH step that branches on an expression, create a list of the conditional steps (target steps) and make them children of the BRANCH step. Then, complete the following:

- In the Properties tab for the BRANCH step, set evaluate-labels to true.
- In the label property of each target, specify the expression that when true, will trigger the target step. The expressions you create can include multiple variables and can specify a range of values for variables. Use the syntax provided by SAP to create the expression. For more information about expression syntax, see Appendix E, “Conditional Expressions” on page 593.

Simple BRANCH Step that branches on an expression

Keep in mind that only one child of a BRANCH step is executed—the target whose label contains an expression that evaluates to true. If none of the expressions evaluate to true, none of the child steps are performed, and execution falls through to the next step in the flow service. You can use the $default value in the label property to designate a default step...
for cases where no expressions evaluate to true. For more information about using the \texttt{Sdefault} setting, see “Specifying a Default Step” on page 114.

\begin{danger}
You cannot branch on switch value and an expression for the same BRANCH step. Branch on the switch value if you want to branch on the value of a single variable and you know the possible run-time values of the switch variable \textit{exactly}. If you want to branch on the values of more than one variable or on a range of values, branch on expressions.
\end{danger}

\begin{danger}
The expressions you create for the children of a BRANCH step need to be mutually exclusive—only one condition should evaluate to true at run time.
\end{danger}

### Specifying a Default Step

If you want to prevent the service from falling through a BRANCH step when an unmatched value occurs at run time, include a default child step to handle unmatched cases. To specify the default alternative of a BRANCH step, set the \texttt{label} property to \texttt{Sdefault}.

The following example shows a BRANCH step that is used to calculate the shipping charges for an order based on the value of a buyer’s account number (\texttt{BuyerAcctNumber}). It contains three target steps. The first two targets single out two accounts for special handling. Note that the labels for these targets specify the account numbers that will trigger them. The third target step has the \texttt{Sdefault} label, so it will process all other accounts.

\begin{danger}
The default step is set to \texttt{Sdefault}.
\end{danger}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{example.png}
\caption{Example of Specifying a Default Step}
\end{figure}

\begin{danger}
You can only have one default target step for a BRANCH step. Developer always evaluates the default step last. Consequently, the default step does not need to be the last child of the BRANCH step.
\end{danger}
Branching on Null and Empty Values

You can use a null variable (a variable that has not been assigned a value or has been explicitly set to null) or an empty string (a string variable that contains no characters) to trigger the execution of a child in a BRANCH step. To branch on one of these values, set the label property for the child step as follows.

<table>
<thead>
<tr>
<th>To BRANCH on...</th>
<th>Do the following...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A null variable</td>
<td>Set the label property to $null.</td>
</tr>
<tr>
<td>An empty string</td>
<td>Leave the label property blank (empty).</td>
</tr>
</tbody>
</table>

Note: You may use $null with any type of switch variable. You may only use an empty value with String variables.

Important! If you are branching on expressions (evaluate-labels is set to true), you cannot branch on null or empty values. Developer ignores children with a blank or $null label.

Using SEQUENCE as the Target of a BRANCH

In many cases, you may want a BRANCH step to conditionally trigger a series of multiple steps rather than just a single step. For these cases, you can use the SEQUENCE step as the target step and group a series of flow steps beneath it.

The following example illustrates a service that accepts a purchase order and processes it one of three ways depending on the payment type specified in the PaymentType variable. Because a series of steps are needed to process the PO in each case, the targets of the BRANCH are defined as SEQUENCE steps, and the appropriate series of steps are specified as children beneath each SEQUENCE.

Use a SEQUENCE step as the target for a multi-step alternative
Define a multi-step alternative in a SEQUENCE

The SEQUENCE step that you use as a target for a BRANCH can contain any valid flow step, including additional BRANCH steps. For additional information about building a SEQUENCE, see “The SEQUENCE Step” on page 125.

Building a BRANCH Step

Use the following procedure to build a BRANCH step in a flow service.

To build a BRANCH step

1. If you are inserting a BRANCH step into an existing flow service, display that service in the Flow Editor and highlight the step immediately above where you want the BRANCH step inserted.

2. Click on the Flow Editor toolbar.

3. Complete the following fields on the Properties tab:

<table>
<thead>
<tr>
<th>For this property...</th>
<th>Specify...</th>
</tr>
</thead>
<tbody>
<tr>
<td>comment</td>
<td>An optional descriptive comment for this step.</td>
</tr>
<tr>
<td>scope</td>
<td>The name of a record in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this field blank.</td>
</tr>
</tbody>
</table>
The BRANCH Step

For this property... Specify...

timeout

The maximum number of seconds that this step should run. When the time is exceeded, the step stops and raises an exception. However, if the BRANCH step contains an INVOKE step, the INVOKE step is not interrupted when the timeout value is exceeded. Instead, the exception is raised after the INVOKE is complete.

If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %expiration%.

If you do not want to specify a timeout period, set timeout to zero or leave it blank.

label

An optional name for this specific step, or a null, unmatched, or empty string ($null, $default, blank). For more information about branching on null or empty values, see “Branching on Null and Empty Values” on page 115.

Note: If you use this step as a target for another BRANCH or an EXIT step, you must specify a value in the label field. For more information about the EXIT step, see “The EXIT Step” on page 130.

switch

The name of the string variable whose value will be used to determine which child step is executed at run time. Do not specify a switch variable if you want to branch on expressions.

evaluate-label

Whether or not you want to branch on expressions. Select true to branch on expressions (evaluate labels as expressions). Select false if you want to branch on the switch value. Default is false.

4 Insert the conditional steps that belong to the BRANCH (i.e., its children) using the following steps:

1 Insert a flow step using the buttons on the Flow Editor toolbar.

2 Indent the flow step using on the Flow Editor toolbar. (Make it a child of the BRANCH step.)
3 In the label field on the Properties tab, specify the switch value that will trigger this step at run time. You may use any of the following as values:

<table>
<thead>
<tr>
<th>Specify...</th>
<th>To match...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A string</td>
<td>That exact string.</td>
</tr>
<tr>
<td>A regular expression</td>
<td>Any string fitting the criteria specified by the regular expression.</td>
</tr>
<tr>
<td>A blank field</td>
<td>An empty string.</td>
</tr>
<tr>
<td>$null</td>
<td>A null string.</td>
</tr>
<tr>
<td>$default</td>
<td>Any unmatched string (i.e., execute the step if the value does not match any other label).</td>
</tr>
</tbody>
</table>

4 Set other properties as needed.

**Important!** If you are branching on expressions, make sure the expressions you assign to the child steps are mutually exclusive.

**Important!** If you are branching on expressions, do not branch on null or empty values. Developer will ignore children with a blank or $null label.

### The REPEAT Step

The REPEAT step allows you to conditionally repeat a sequence of child steps based on the success or failure of those steps. You can use REPEAT to:

- **Re-execute (retry) a set of steps if any step within the set fails.** This option is useful to accommodate transient failures that might occur when accessing an external system (e.g., databases, ERP systems, or Web servers) or device.

- **Re-execute a set of steps until one of the steps within the set fails.** This option is useful for repeating a process as long as a particular set of circumstances exists (e.g., data items exist in a data set).
Use REPEAT to re-execute one or more steps

This INVOKE step is repeated up to 10 times if it fails at run time.

Specifying the REPEAT Condition

When you build a REPEAT step, you set the `repeat-on` property to specify the condition—success or failure—that will cause its children to re-execute at run time.

If you set “repeat on” to...

<table>
<thead>
<tr>
<th>Value</th>
<th>The REPEAT step...</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAILURE</td>
<td>Re-executes the set of child steps if any step in the set fails.</td>
</tr>
<tr>
<td>SUCCESS</td>
<td>Re-executes the set of child steps if all steps in the set complete successfully.</td>
</tr>
</tbody>
</table>
Setting the REPEAT Counter

The REPEAT step’s count property determines whether the REPEAT step is limited to a specified number of repetitions, and if so, what that limit is.

<table>
<thead>
<tr>
<th>If you set the “count” to...</th>
<th>The REPEAT step...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Does not re-execute children.</td>
</tr>
<tr>
<td>Any value &gt; 0</td>
<td>Re-executes children up to this number of times.</td>
</tr>
<tr>
<td>-1</td>
<td>Re-executes children as long as the specified repeat condition is true.</td>
</tr>
</tbody>
</table>

**Important!** Note that children of a REPEAT always execute at least once. The count property specifies the maximum number of times the children can be re-executed. At the end of an iteration, the server checks to see whether the condition (i.e., failure or success) for repeating is satisfied. If the condition is true and the count is not met, the children are executed again. This process continues until the repeat condition is false or count is met, whichever occurs first. (In other words, the maximum number of times that children of a REPEAT will execute when count is > -1, is count+1.)

When Does REPEAT Fail?

The following conditions cause the REPEAT step to fail:

<table>
<thead>
<tr>
<th>If “repeat on” is set to...</th>
<th>The REPEAT step fails if...</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>A child within the REPEAT block fails.</td>
</tr>
<tr>
<td>FAILURE</td>
<td>The count limit is reached before its children execute successfully.</td>
</tr>
</tbody>
</table>

If the REPEAT step is a child of another flow step, the failure is propagated to its parent.

Using REPEAT to Retry a Failed Step

If your flow invokes services that access external systems, you can use the REPEAT step to accommodate network errors—busy servers, connection errors, and so forth—at run time. If you use the REPEAT step for this purpose, keep the following points in mind:

- The following types of failures satisfy the FAILURE condition:
  - Expiration of a child step’s timeout limit.
  - An exception thrown by a Java service
  - A document query that returns an unpermitted null value.
If you specify multiple children under a REPEAT step, the failure of any one of the children will cause the entire set of children to be re-executed.

The REPEAT step immediately exits a set of children at the point of failure (i.e., if the second child in a set of three fails, the third child is not executed).

When repeat-on is set to failure, the failure of a child within a REPEAT step does not cause the REPEAT step itself to fail unless the count limit is also reached.

The timeout property for the REPEAT step specifies the amount of time in which the entire REPEAT step, including all of its possible iterations, must complete. When you use REPEAT to retry on failure, you may want to leave the timeout value at 0 (no limit) or set it to a very high value. You can also set the property to the value of a pipeline variable by typing the name of the variable between % symbols.

As a developer, you must be thoroughly familiar with the processes you include within a REPEAT step. Make certain that the child steps you specify can safely be repeated in the event that a failure occurs. You don’t want to use REPEAT if there is the possibility that a singular action—such as accepting an order or crediting an account balance—could be applied twice.

To build a REPEAT step that re-executes failed steps

1. If you are inserting a REPEAT step into an existing flow service, display that service in the Flow Editor and highlight the step immediately above where you want the REPEAT step inserted.

2. Click the Flow Editor toolbar.

3. Complete the following fields on the Properties tab:

<table>
<thead>
<tr>
<th>For this property...</th>
<th>Specify...</th>
</tr>
</thead>
<tbody>
<tr>
<td>comment</td>
<td>An optional descriptive comment for this step.</td>
</tr>
<tr>
<td>scope</td>
<td>The name of a record in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this field blank.</td>
</tr>
</tbody>
</table>
For this property... Specify...

**timeout**

The maximum number of seconds that this step should run. When the time is exceeded, the step stops and raises an exception. However, if this step contains an INVOKE step, the INVOKE step is not interrupted when the timeout value is exceeded. Instead, the exception is raised after the INVOKE is complete.

If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %expiration%.

If you do not want to specify a timeout period, set **timeout** to zero or leave it blank.

**label**

An optional name for this specific REPEAT step, or a null, unmatched, or empty string ($null, $default, blank).

**Important!** If you use this step as a target for a BRANCH or EXIT step, you **must** specify a value in the **label** field. For more information about the BRANCH and EXIT steps, see “The BRANCH Step” on page 110 or “The EXIT Step” on page 130.

**count**

The maximum number of times you want the children to be re-executed. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %servicecount%.

If you want the children re-executed until they are all successful (i.e., no maximum limit), set this value to –1.

**backoff**

The length of time (in seconds) that you want the server to wait between iterations of the children.

If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %waittime%.

**repeat on**

FAILURE

4 Beneath the REPEAT step, use the following steps to insert each step that you want to repeat:

1 Insert a step using the buttons on the Flow Editor toolbar.

2 Indent that step using on the Flow Editor toolbar. (Make it a child of the REPEAT step.)

3 Set the properties for the child step as needed.
Using REPEAT to Retry a Successful Step

Apart from using REPEAT to retry a failed step, you can also use it as a looping device to repeat a series of steps until a failure occurs. In a Web-automation service, you could use it to repeat a load and query step until it fails to produce a required value. For example, you might repeat a set of steps as long as the query finds a “Next Page” button in the current document, indicating that there are additional pages to be processed. In this situation, you would want to end the loop when the query step “fails” to retrieve a “Next Page” button in the current document.

If you use the REPEAT step to re-execute successful child steps, keep the following points in mind:

- The success condition is met if all children of the REPEAT step execute without returning a single exception.
- If one child in the set fails, the REPEAT step exits at the point of failure, leaving the remaining children unexecuted.
- The failure of a child does not cause the REPEAT step to fail; it merely ends the loop. (In this case, the REPEAT step itself succeeds and execution of the flow proceeds normally).

To build a REPEAT step that repeats a set of successful steps

1. If you are inserting a REPEAT step into an existing flow service, display that service in the Flow Editor and highlight the step immediately above where you want the REPEAT step inserted.

2. Click on the Flow Editor toolbar.

3. Complete the following fields on the Properties tab:

<table>
<thead>
<tr>
<th>For this property...</th>
<th>Specify...</th>
</tr>
</thead>
<tbody>
<tr>
<td>comment</td>
<td>An optional descriptive comment for this step.</td>
</tr>
<tr>
<td>scope</td>
<td>The name of a record in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this field blank.</td>
</tr>
</tbody>
</table>
Beneath the REPEAT step, use the following steps to insert each step that you want repeat:

1. Insert a flow step using the buttons on the Flow Editor toolbar.
2. Indent that flow step using \( \text{F3} \) on the Flow Editor toolbar. (Make it a child of the REPEAT step.)
3. Set the properties for the child step as needed.

<table>
<thead>
<tr>
<th>For this property...</th>
<th>Specify...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>timeout</strong></td>
<td>The maximum number of seconds that this step should run. When the time is exceeded, the step stops and raises an exception. However, if this step contains an INVOKE step, the INVOKE step is not interrupted when the timeout value is exceeded. Instead, the exception is raised after the INVOKE is complete. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %expiration%. If you do not want to specify a timeout period, set timeout to zero or leave it blank.</td>
</tr>
<tr>
<td><strong>label</strong></td>
<td>An optional name for this specific step, or a null, unmatched, or empty string ($null, $default, blank). <strong>Important!</strong> If you use this step as a target for a BRANCH or EXIT step, you must specify a value in the label field. For more information about the BRANCH and EXIT steps, see “The BRANCH Step” on page 110 or “The EXIT Step” on page 130.</td>
</tr>
<tr>
<td><strong>count</strong></td>
<td>The maximum number of times you want the children to be re-executed. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %servicecount%. If you want the children re-executed until any one of them fails (i.e., no maximum limit), set this value to –1.</td>
</tr>
<tr>
<td><strong>backoff</strong></td>
<td>The length of time (in seconds) that you want the server to wait between iterations of the children. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %waittime%.</td>
</tr>
</tbody>
</table>

4. Beneath the REPEAT step, use the following steps to insert each step that you want repeat:
The SEQUENCE Step

You use the SEQUENCE step to build a set of steps that you want to treat as a group. Steps in a group are executed in order, one after another. By default, all steps in a flow service, except for children of a BRANCH step, are executed as though they were members of an implicit SEQUENCE step—i.e., they execute in order, one after another. However, there are times when it is useful to explicitly group a set of steps. The most common reasons to do this are:

- To group a set of steps as a single alternative beneath a BRANCH step. For details about this use of the SEQUENCE step, see “Using SEQUENCE as the Target of a BRANCH” on page 115.
- To specify the conditions under which the server will exit a sequence of steps without executing the entire set.

Using SEQUENCE to Specify an Exit Condition

In an implicit sequence, when a step fails, the server automatically exits the sequence (i.e., the exit-on property is set to FAILURE). By grouping steps into an explicit sequence, you can override this default behavior, and specify the condition on which the sequence exits. To do this, you set the exit-on parameter as follows:

Set exit-on to... If you want the server to...

**FAILURE**

Exit the sequence when a step in the sequence fails. (Execution continues with the next flow step in the flow service.) This is the default behavior of a sequence of steps.

This setting is useful if you have a series of steps that build upon one another. For example, if you have a set of steps that gets an authorization code and then submits a PO, you will want to skip the PO submission if the authorization step fails.

When a SEQUENCE exits under this condition, the SEQUENCE step fails.

**SUCCESS**

Exit the sequence when any step in the sequence succeeds. (Execution continues with the next step in the flow service.)

This setting is useful for building a set of alternative steps that are each attempted at run time. Once one of the members of the set runs successfully, the remaining steps in the sequence are skipped.

When a SEQUENCE exits under this condition, the server considers the SEQUENCE step successful. The only time it fails is if all its children fail.
Set exit-on to... If you want the server to...

DONE

Execute every step in the sequence even if one of the steps in the sequence fails.

The server considers a SEQUENCE step successful as long as it executes all of its children within the specified timeout limit. The success or failure of a child within the sequence is not taken into consideration. If a child fails under this condition, any changes that it made to the pipeline are rolled back (undone), and processing continues with the next child step in the SEQUENCE.

The LOOP Step

The LOOP step repeats a sequence of child steps once for each element in an array that you specify. For example, if your pipeline contains an array of purchase-order line items, you could use a LOOP step to process each line item in the array.

To specify the sequence of steps that make up the body of the loop (i.e., the set of steps you want the LOOP to repeat), you indent those steps beneath the LOOP as shown in the following example.

Simple LOOP Step

You may include any valid flow step within the body of a LOOP, including additional LOOP steps. The following example shows a pair of nested LOOPS. Note how the indentation of the steps determines the LOOP to which they belong.
Nested LOOP Steps

The entire LOOP step is a child of the outer LOOP.

Specifying the Input Array

The LOOP step requires you to specify an input array that contains the individual elements that will be used as input to one or more steps in the LOOP. At run time, the LOOP step executes one pass of the loop for each member in the specified array. For example, if you want to execute a LOOP for each line item stored in a purchase order, you would use the Record List in which the order’s line items are stored as the LOOP’s input array.

You specify the name of the input array on the LOOP step’s Properties tab. The array you specify can be any of the following data types:

- String List
- String Table
- Record List
- Object List
When you design your flow, bear in mind that because the services within the loop operate against individual elements in the specified input array, they must be designed to take elements of the array as input, not the entire array.

For example, if your LOOP executes against a Record List called LineItems that contains children called Item, Qty, and UnitPrice, you would specify LineItems as the in-array for the LOOP step, but services within the loop would take the individual elements of LineItems (e.g., Item, Qty, UnitPrice, and so forth) as input.

### Collecting Output from a LOOP Step

If your LOOP step produces an output variable, the server can collect that output into an array in the pipeline.

To do this, you use the out-array parameter to specify the name of the array variable into which you want the server to collect output for each iteration of the loop. For example, if your loop checks inventory status of each line item in a purchase order and produces a String called InventoryStatus each time it executes, you would specify InventoryStatus as the value of out-array. At run time, the server will automatically transform InventoryStatus to an array variable that contains the output from each iteration of the loop.

To collect output from each pass of the loop, specify the name of the output variable that you want the server to collect for each iteration.
Building a LOOP Step

Use the following procedure to build a LOOP step in a flow service.

To build a LOOP step

1. If you are inserting a LOOP step into an existing flow service, display that service in the Flow Editor and select the step immediately above where you want the LOOP step inserted.

2. Click on the Flow Editor toolbar.

3. Complete the following fields on the Properties tab:

<table>
<thead>
<tr>
<th>For this property...</th>
<th>Specify...</th>
</tr>
</thead>
<tbody>
<tr>
<td>comment</td>
<td>An optional descriptive comment for this step.</td>
</tr>
<tr>
<td>scope</td>
<td>The name of a record in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this field blank.</td>
</tr>
<tr>
<td>timeout</td>
<td>The maximum number of seconds that this step should run. When the time is exceeded, the step stops and raises an exception. However, if this step contains an INVOKE step, the INVOKE step is not interrupted when the timeout value is exceeded. Instead, the exception is raised after the INVOKE is complete. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %expiration%. If you do not want to specify a timeout period, set timeout to zero or leave it blank.</td>
</tr>
<tr>
<td>label</td>
<td>An optional name for this specific LOOP step, or a null, unmatched, or empty string ($null, $default, blank). Important! If you use this step as a target for a BRANCH or EXIT step, you must specify a value in the label field. For more information about the BRANCH and EXIT steps, see “The BRANCH Step” on page 1100 or “The EXIT Step” on page 130.</td>
</tr>
</tbody>
</table>
CHAPTER 5 Inserting Flow Steps

For this property... Specify...
in-array The name of the array variable on which the LOOP will operate. This variable must be one of the following types: String List, String Table, Record List, Object List.
out-array The name of the element that you want the server to collect each time the LOOP executes. You do not need to specify this property if the loop does not produce output values or if you are collecting the elements of in-array.

4 Build the body of the loop using the following steps:
1 Insert a flow step using the buttons on the Flow Editor toolbar.
2 Indent the flow step using \[ \text{_indent} \] on the Flow Editor toolbar. (Make it a child of the LOOP step.)
3 Set the properties for the child step as needed.
5 Use the Pipeline Editor to map the elements of the input array to the input variables required by each child of the LOOP step. For more information about using the Pipeline Editor, see “Mapping Data in a Flow Service” on page 149.

The EXIT Step

The EXIT flow step allows you to exit the entire flow service or a single flow step. You specify whether you want to exit from:
- The nearest ancestor LOOP or REPEAT flow step to the EXIT flow step.
- The parent flow step of the EXIT flow step.
- A specified ancestor flow step to the EXIT flow step.
- The entire flow service.

When you use the EXIT step, you indicate whether exiting should return a successful condition or a failure condition. If the exit is considered a failure, an exception is thrown. You can specify the text of the error message that is displayed by typing it directly or by assigning it to a variable in the pipeline.

Examples of when to use the EXIT step include to:
- Exit an entire flow service from within a series of deeply nested steps.
- Throw an exception when you exit a flow or a flow step without having to write a Java service to call Service.throwError( ).
- Exit a LOOP or REPEAT flow step without throwing an exception.
To build an EXIT step

1. If you are inserting an EXIT step into an existing flow service, display that service in the Flow Editor and select the step immediately above where you want the EXIT step inserted.

2. Click on the Flow Editor toolbar.

3. Complete the following fields on the Properties tab:

   For this property... | Specify...
   ---------------------|-------------------
   comment              | An optional descriptive comment for this step.
   label                | An optional name for this specific step, or a null, unmatched, or empty string ($null, $default, blank).

   **Important!** If you use this step as a target for a BRANCH step, you **must** specify a value in the label field. For more information about the BRANCH step, see “The BRANCH Step” on page 110.

   from
   The flow step that you want to exit from. Specify one of the following:

   Specify | To exit from the...
   --------|-------------------
   $loop   | Nearest ancestor LOOP or REPEAT flow step.
   $parent | Parent flow step, regardless of the type of step.
   $flow   | Entire flow.
   label   | Nearest ancestor flow step that has a label that matches this value.

   **Note:** If the label you specify does not match the label of an ancestor flow step, the flow will exit with an exception.

   signal
   Whether the exit is to be considered a success or a failure. Specify one of the following:

   Specify... | To...
   ----------|-------
   SUCCESS   | Exit the flow service or flow step with a success condition.
The MAP Step

The MAP step lets you adjust the contents of the pipeline at any point in a flow service. When you build a MAP step, you can:

- Prepare the pipeline for use by a subsequent step in the flow service by mapping, adding, and dropping variables in the pipeline.
- Clean up the pipeline after a preceding step, by removing fields that the step added but are not needed by subsequent steps.
- Move variables or assign values to variables in the pipeline.
- Initialize the input values for a flow service.
- Invoke several services (transformers) in a single step.

Tip! The MAP step is especially useful for hard coding an initial set of input values in a flow service. To use it in this way, insert the MAP step at the beginning of your flow, and then use the Set Value modifier to assign values to the appropriate variables in Pipeline Out.

For more information about the MAP step, see “Mapping Data in a Flow Service” on page 149.
Using Flow Diagram View

- What Does a Flow Service Look Like in Diagram View? .................. 136
- Switching to Flow Diagram View ....................................................... 140
- Building a Flow Service in Diagram View ........................................ 140
- Changing the Layout of a Flow Service ........................................... 147
What Is Flow Diagram View?

Flow diagram view, like flow tree view, is a view of a flow service that Developer displays in the Flow Editor. You can use either view to build or edit a flow service. However, diagram view provides a more graphical view in which to create flow services.

In diagram view, a flow service looks similar to a flow chart—Developer displays shapes for flow steps as well as for the start and end of the flow service. Developer uses lines to connect the flow steps and to show the order in which the flow steps execute.

The following illustration shows a flow service in diagram view. Notice the layout of the flow steps in the Flow Editor.

Flow service in flow diagram view
What Is Flow Diagram View?

Following is the same flow service in tree view.

Flow service in flow tree view

![Flow service in flow tree view](image)

Notice that tree view provides a condensed view of the flow service. Developer lists flow steps sequentially, top to bottom, and executes the steps in that order.

Note: You can use diagram view to view and edit any flow service that you created in an earlier version of SAP BC Server.

Note: Developer uses tree view as the default view for flow services. With the exception of the procedures in this chapter, the procedures in this book are written for working in tree view.

When Should You Use Flow Diagram View?

Because diagram view and tree view provide the same capabilities for building a flow service, work in whichever view you find easier to use. You can easily switch between views when building a flow service. For example, you might find it easier to insert the flow steps and define the basic structure of a flow service in diagram view, but use tree view to perform data mapping.

You might prefer to use diagram view if:

- You find that building a flow service as a flow chart is easier than building a flow service as a sequence of statements. You might be able to more easily envision the processes a flow service performs if you view the flow service as a diagram instead of as a series of line-by-line steps.
- You need to design a business process with someone unfamiliar with programming or unfamiliar with SAP BC Server. People who are not familiar with programming might be more comfortable with flow charts.
You need to show diagrams of how the flow service works to management. (Flow services can be exported to HTML and then printed.)

You want to be able to customize the appearance of the flow service without changing the order in which flow steps are executed. In diagram view, you can place the flow steps in an arrangement that you want. Developer maintains the lines between the flow steps and the structure of the flow service.

You want to use Trace to debug a flow step an operation. It might be easier to follow the path of execution when Developer presents the flow as a diagram.

What Does a Flow Service Look Like in Diagram View?

Diagram view uses specific shapes and structures for the elements of a flow service, such as the start and end of a flow service, parent steps, child steps, and the order in which flow steps run.

The following illustration identifies the basic elements of a flow service in diagram view.
What Does a Flow Service Look Like in Diagram View?

Basic elements of a flow service in flow diagram view

developer automatically inserts the start and end symbols when you create a flow service. When you insert a step into a flow service, developer automatically draws the lines connecting the flow step to the rest of the steps in the service.

Note: Developer automatically draws, redraws, and deletes lines when you insert, move, or delete steps in a flow service. You cannot move or delete lines.

Steps such as BRANCH, LOOP, and REPEAT can contain child steps. By default, Developer displays these steps collapsed—you cannot see the individual child steps. You can expand these steps. When you do, Developer uses a solid colored line to indicate the path of data within the flow step. Developer also uses dashed lines to enclose the children under the parent step. Developer places a small rectangle after the last child step to indicate the end of the parent step.
Child steps in a flow service in flow diagram view

![Image of flow diagram view](image)

- **Dashed lines** enclose the children of a step.
- **Solid lines** indicate the path of data within the parent step.
- **Indicates the end of the parent step.**
- **Click to expand and view the children of the step.**
- **Click to collapse and hide the children of the step.**

**Note:** Notice that lines enclosing the child steps appear in the same color as the parent step symbol. Developer also uses the same color around the small rectangle that it places after the last child step.

In diagram view, the rectangle that contains the flow step displays properties for the step, such as **label** and **comment**. Each rectangle also displays an additional property that is relevant to the flow step type, such as **in-array** for LOOP and **switch** for BRANCH.
Flow step properties in a flow service

The following table indicates which property is shown for each flow step.

<table>
<thead>
<tr>
<th>This step...</th>
<th>Displays this property...</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRANCH</td>
<td>switch specifies the name of the variable whose value triggers the execution of one of the BRANCH step’s children at run time. If you branch on expressions, this property is blank.</td>
</tr>
<tr>
<td>EXIT</td>
<td>from specifies the label of the nearest ancestor flow step from which you want to exit.</td>
</tr>
<tr>
<td>INVOKE</td>
<td>service specifies the name of the service that is invoked at run time.</td>
</tr>
<tr>
<td>LOOP</td>
<td>in-array specifies the name of the array against which the selected LOOP step will run. Type the name of this variable exactly as it will appear in the pipeline at run time.</td>
</tr>
<tr>
<td>MAP</td>
<td>none</td>
</tr>
<tr>
<td>REPEAT</td>
<td>none</td>
</tr>
<tr>
<td>SEQUENCE</td>
<td>none</td>
</tr>
</tbody>
</table>
CHAPTER 6  Using Flow Diagram View

Switching to Flow Diagram View

When working with a flow service, you can switch between tree view and diagram view.

To switch to Flow Diagram view

- Click on the Flow Editor toolbar.
- OR–
- On the View menu, select Diagram View.

To switch to Flow Tree view

- Click on the Flow Editor toolbar.
- OR–
- On the View menu, select Tree View.

Building a Flow Service in Diagram View

Building a flow service in diagram view consists of the same stages as building a flow service in tree view—you need to create the flow service, insert flow steps, set properties, declare the service input and output parameters, map pipeline data, and set run-time parameters. With the exception of how you insert and move flow steps, the procedures for completing each stage are the same in diagram view as they are in tree view.

Note: You might find it easier to build services in diagram view if you have larger view of the Flow Editor. To maximize the size of the Flow Editor, click anywhere in the Flow Editor, and click on the toolbar.

Inserting a Flow Step

When you select the flow step you want to insert, Developer displays “drop zones” that indicate where you can insert the step. Drop zones appear before and after each step in the flow service.

When you move the mouse over a drop zone, a green circle surrounds the drop zone. To place a flow step, you click the drop zone where you want to insert the step. You can insert a step at any drop zone in the flow service. The following illustration shows the drop zones for a flow service.
Drop zones appear before and after steps.

A green circle surrounds the selected drop zone.

**Note:** The following procedure provides general information for inserting steps in diagram view. For specific information about working with BRANCH steps in diagram view, see “Inserting a Child Step into a BRANCH Step” on page 142.
To insert a flow step

1. In the Flow Editor, display the service in which you want to insert a step. Make sure that you are in Flow Diagram view.

2. On the Flow Editor toolbar, click the button representing the step you want to insert. For example, if you are inserting an INVOKE step, click \( \text{button} \), and select the service you want to invoke.

   Developer displays all of the drop zones for the flow service. To view the drop zones in a collapsed flow step, move the mouse pointer over the \( \text{drop zone} \) in the flow step rectangle.

3. Click the drop zone where you want to insert the flow step.

   Developer inserts the step and automatically draws lines to connect the step to the rest of the steps in the flow service.

4. Click \( \text{button} \) to save your changes to the server.

Note: To cancel the insertion of a flow step before you select a drop zone, click \( \text{button} \) on the Flow Editor toolbar.

Inserting a Child Step into a BRANCH Step

When you build a BRANCH step in diagram view, you can

- Insert a flow step as a child (target) of the BRANCH, or
- Insert a flow step into an existing target

The locations of the drop zones within a BRANCH step indicate where you can place flow steps. (Drop zones only appear when you insert or relocate a step.) The following illustration displays the drop zones inside a BRANCH step.
Drop zones for a BRANCH step

The drop zones that appear on the horizontal line above the children (targets) of the BRANCH step indicate locations for new targets. The drop zones that appear on the lines to and from the targets of a BRANCH indicate locations where a flow step can be inserted into an existing target.

Note: In diagram view, Developer evaluates the target steps of a BRANCH from left to right—Developer evaluates the label of the left-most target first. (In tree view, Developer evaluates the target steps from top to bottom.) At run time, Developer executes the first target with a matching switch value or an expression that evaluates to true. A target with the $default label will be evaluated last, regardless of its position.

Inserting an Step into an Existing Target

As indicated in the preceding illustration, you can insert an step into an existing target. A target of a BRANCH needs to be a single flow step. (This step can contain child flow steps.) Therefore, if you insert an step into an existing target, Developer automatically combines the existing step and the inserted step into a single flow step. The resulting flow step becomes the target of the BRANCH.

The flow step that Developer generates depends on the type of step you insert into the target and where you insert it:

- If you insert an INVOKE, MAP, or EXIT step above the existing step in a target, Developer combines the steps into a new SEQUENCE step.
If you insert a container step (a flow step that can contain children) above the existing step in a target, Developer uses the container step as the target of the BRANCH. Developer makes the existing target step the last child of the inserted container step.

If you insert any type of flow step below the existing step in a target, Developer combines the inserted and existing flow steps into a new SEQUENCE step.

For more information about the flow step that SAP BC generates when you insert steps into an existing target of a BRANCH, see the following sections.

**Note:** The behavior described in the following sections is unique to diagram view and does not occur in tree view.

**Inserting an INVOKE, MAP, or EXIT Step Above an Existing Target**

When you insert an INVOKE, MAP, or EXIT step above the existing target step, Developer automatically wraps both steps with a new SEQUENCE step. The SEQUENCE becomes the target of the BRANCH step. Developer leaves the label of the SEQUENCE blank.

**Inserting a MAP or INVOKE before the existing target step**
### Important!
Make sure that you assign a label to the new SEQUENCE step. For more information about assigning labels to the children of BRANCH steps, see “The BRANCH Step” on page 110.

---

**Inserting a Container Step Above an Existing Target**

When you insert or move a container step (one that can contain children, such as REPEAT or LOOP) before the existing target step, Developer automatically makes the existing target step the last child of the container step. The container step becomes the target of the BRANCH. Developer does not change the label of the container step.

### Inserting a container step before the existing target step

Important! Make sure that you assign a label to the container step is the label you want to use for the target step. For more information about assigning labels to the children of BRANCH steps, see “The BRANCH Step” on page 110.
Inserting a Flow Step Below an Existing Target

When you insert or move a flow step after an existing target step, Developer automatically wraps both steps with a SEQUENCE step. The SEQUENCE becomes the target of the BRANCH step. Developer leaves the label of the SEQUENCE blank.

Inserting an step after the existing target step

If you insert a step here...

...SAP BC Developer wraps a SEQUENCE around both steps.

![Diagram of a process flow with a SEQUENCE step added after an existing target step.]

**Important!** Make sure that you assign a label to the new SEQUENCE step. For more information about assigning labels to the children of BRANCH steps, see “The BRANCH Step” on page 110.
Changing the Order of Steps in a Flow Service

In diagram view, the arrows connecting the flow steps indicate the order in which the steps execute. You can move, or relocate, steps in a flow service to change the order in which steps execute. You can also relocate a step to make it a child of another step in the flow service.

**To relocate a step**

1. In the Flow Editor, select the flow step you want to relocate.
2. On the Edit menu, select **Relocate**. Developer removes the step from the flow service and displays the drop zones where you can place the step.

   To view the drop zones in a collapsed step, move the mouse pointer over the 📡 in the step rectangle.

**Note:** You can also press ALT+R to remove the selected step from the flow service and display the drop zones. (This shortcut key does not work if you are running Developer on a Solaris platform.)

3. Click the drop zone where you want to move the step.
4. Click 📡 to save your changes to the server.

**Note:** To cancel relocation of a step before you select a drop zone, click 🗑 on the Flow Editor toolbar.

You can also change the order in which flow steps run in a flow service by cutting, copying, and pasting steps. For step-by-step procedures for cutting, copying, and pasting steps in diagram view, see online help.

Changing the Layout of a Flow Service

A significant difference between diagram view and tree view is that in diagram view, you can change the layout of a flow service without changing the logic of the flow service. By changing the layout of steps in a flow service, you might be able to more clearly express the sequence of steps and the flow of data. You can create the layout that you think best illustrates the work performed by the flow service.

**Note:** When you change the layout of steps in a flow service, you are only changing the appearance of the flow service in diagram view—you are not changing the actual logic of the flow service.
To change the layout of a flow service

1. Select the flow step you want to move and drag it to a new location in the Flow Editor. Developer automatically adjusts the flow lines when you move the flow step.

2. Click to save your changes. Developer displays the new layout the next time you open the flow service.

Note: Use the and buttons on the Flow Editor toolbar to zoom in on or zoom out of the flow service.

Resetting the Flow Service Layout

You can reset a flow service to its default layout. In the default layout, Developer aligns all the steps in the flow service at the left edge of the Flow Editor and lists the steps sequentially. You might want to return a flow service to the default layout if the current layout is too confusing or too cluttered, or if flow steps are too far apart in the Flow Editor.

To reset the flow service to the default layout

- Click on the Flow Editor toolbar.

  Developer aligns the flow steps at the left edge of the Flow Editor. Steps retain their expanded or collapsed state when you reset the flow service.
Mapping Data in a Flow Service

- What Is the Pipeline Editor? ................................................................. 150
- Basic Mapping Tasks ................................................................. 154
- Working with Transformers .................................................. 177
Mapping is the process of performing transformations to resolve data representation differences between services or document formats. By mapping, you can accomplish the following types of data transformations:

- **Name transformations** where different variable names represent the same data item. For example, one service or document format might use *Telephone* as the name of the variable for telephone number information and another might use *PhoneNumber*. When you perform name transformations, the value and position of a variable in the record structure remain the same, but the name of the variable changes.

- **Structural transformations** where different data structures represent a data item. For example, one service or document format might put the telephone number in a string called *Telephone*, and the next may expect to find it in an element of a record array called *CustInfo*. When you perform structural transformations, the value of the variable remains the same, but the data type or position of the variable in the record structure changes.

- **Value transformations** where different formats represent the same value. This occurs commonly with date and time variables, where, for instance, one variable might use “01/01/99” and another “January 1, 1999.” In other cases, your services or document formats might use different notations for standard codes or values, different currency units, or a different system of weights and measures (metric instead of the U.S. Customary or British Imperial systems). When you perform value transformations, the name and position of the variable remain the same, but the data contained in the variable changes. (For example, you can change the format of a date, concatenate two strings, or add the values of two variables together.)

When you build flow services or convert between document formats, you may need to perform one, two, or all of the above types of data transformation. The webMethods flow language provides two ways for you to accomplish data transformations between services and document formats—you can map variables to each other (create links) or you can insert transformers.

SAP BC Developer provides a graphical environment in which you can perform mapping between variables and formats—the Pipeline Editor.

**What Is the Pipeline Editor?**

The Pipeline Editor is the facility you use to perform mapping and inspect the pipeline. It offers a graphical representation of all of your data. You use the tools in the Pipeline Editor to route variables (data) between services or between document formats.

Access the Pipeline Editor by selecting the **Pipeline** tab for an invoked service (INVOKE step) or a MAP step in a flow service. The Pipeline Editor contents for INVOKE steps are slightly different than the contents for MAP steps.
Pipeline Editor for an INVOKE Step

For an INVOKE step, the Pipeline Editor depicts two stages of the pipeline with respect to the selected service in the Flow Editor.

Pipeline Editor for an INVOKE Step (service)

The Pipeline tab depicts the service's input and output with respect to the expected pipeline.

<table>
<thead>
<tr>
<th>Pipeline In</th>
<th>Device In</th>
<th>Service Out</th>
<th>Pipeline Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcctNumber</td>
<td>S@APBC</td>
<td>T@N@X</td>
<td>AcctNumber</td>
</tr>
<tr>
<td>DocType</td>
<td>P@N</td>
<td>T@N@X</td>
<td>DocType</td>
</tr>
<tr>
<td>PONumber</td>
<td>T@N@X</td>
<td>T@N@X</td>
<td>PONumber</td>
</tr>
<tr>
<td>Date</td>
<td>T@N@X</td>
<td>Date</td>
<td>Date</td>
</tr>
<tr>
<td>OrderNo</td>
<td>T@N@X</td>
<td>OrderNo</td>
<td>OrderNo</td>
</tr>
<tr>
<td>OrderDate</td>
<td>T@N@X</td>
<td>OrderDate</td>
<td>OrderDate</td>
</tr>
<tr>
<td>Supplier</td>
<td>T@N@X</td>
<td>Supplier</td>
<td>Supplier</td>
</tr>
<tr>
<td>OrderDate</td>
<td>T@N@X</td>
<td>OrderDate</td>
<td>OrderDate</td>
</tr>
</tbody>
</table>

This stage... Represents...

1. The expected state of the pipeline just before the selected service executes.

   **Pipeline In** depicts the set of variables that are expected to be in the pipeline before the service executes (based on the declared input and output parameters of the preceding services).

   **Service In** depicts the set of variables the selected service expects as input (as defined by its input parameters).

Using the Pipeline Editor, you can insert “pipeline modifiers” at this stage to adjust the contents of the pipeline to suit the requirements of the service. For example, you can map variables, assign values to variables, drop variables from the pipeline, or add variables to the pipeline. Modifications that you specify during this stage are performed immediately before the service executes at run time.
CHAPTER 7 Mapping Data in a Flow Service

Pipeline Editor for a MAP Step

For a MAP step, the Pipeline Editor displays a single stage of the pipeline. The Pipeline Editor displays two sets of variables: **Pipeline In** and **Pipeline Out**. Between these sets of variables, the Pipeline Editor displays a column named **Transformers**.

### Pipeline Editor for a MAP Step

The expected state of the pipeline just after the service executes. **Service Out** depicts the set of variables that the selected service produces as output (as defined by its output parameters).

**Pipeline Out** depicts the set of variables that are expected to be in the pipeline after the service executes. It represents the set of variables that will be available to the next service in the flow. If the selected service (INVOKE step) is the last step in the flow service, **Pipeline Out** displays the output variables for the flow service (as declared on the **Input/Output** tab).

Using the Pipeline Editor, you can insert “pipeline modifiers” at this stage to adjust the contents of the pipeline. For example, you can map variables, assign values to variables, drop variables from the pipeline, or add variables to the pipeline. Modifications that you specify during this stage are performed immediately after the service executes at run time.
The **Pipeline In** column represents input to the MAP step. It contains the names all of the variables in the pipeline at this point in the flow.

The **Transformers** column displays any services inserted in the MAP step to complete value transformations. For more information about invoking services in a MAP step, see “Inserting a Transformer into a MAP Step” on page 179.

The **Pipeline Out** column represents the output of the MAP step. It contains the names of variables that will be available in the pipeline when the MAP step completes.

When you first insert a MAP step into your flow, **Pipeline In** and **Pipeline Out** are identical. However, if the MAP step is the only step in the flow service or is the last step in the flow service, **Pipeline Out** also displays the variables declared as output in the flow service.

Using the Pipeline Editor, you can insert “pipeline modifiers” to adjust the contents of the pipeline. For example, you can map variables from **Pipeline In** to services in **Transformers**. You can also use pipeline modifiers to assign values to pipeline variables, drop variables from the pipeline, or add variables to the pipeline. Use pipeline modifiers to map variables to each other or to adjust the contents of the variables in the Pipeline Editor.

### Pipeline Modifiers

Pipeline modifiers are special commands that you apply to adjust the pipeline at run time. They execute immediately before or after the selected service or transformer, depending on where you add them to your pipeline map. Use the following buttons to add pipeline modifiers to the pipeline:

<table>
<thead>
<tr>
<th>Use this modifier...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Map" /> <strong>Map</strong></td>
<td><strong>Map a pipeline variable to a service variable.</strong> The <strong>Map</strong> modifier lets you resolve variable-name and data-structure differences by “mapping” (copying) the value of one variable to another at run time. For information about using this pipeline modifier, see “Mapping Variables” on page 155.</td>
</tr>
<tr>
<td><img src="image" alt="Drop" /> <strong>Drop</strong></td>
<td><strong>Drop a variable from the pipeline.</strong> The <strong>Drop</strong> modifier removes extraneous variables from the pipeline. For information about using this pipeline modifier, see “Dropping Variables from the Pipeline” on page 175.</td>
</tr>
<tr>
<td><img src="image" alt="Set Value" /> <strong>Set Value</strong></td>
<td><strong>Assign a value to a variable.</strong> The <strong>Set Value</strong> modifier “hard codes” a value for a variable. For information about this pipeline modifier, see “Assigning Values to Pipeline Variables” on page 172.</td>
</tr>
</tbody>
</table>
Printing the Pipeline Editor

The following procedure describes how to use the View as HTML command to produce a printable version of the Pipeline Editor.

**Note:** When you view the pipeline editor as HTML, the resulting HTML page displays only the portion of the pipeline editor that is visible on the Pipeline tab. Before you select the View as HTML command, make sure the Pipeline tab displays the part of the pipeline editor that you want to view as HTML.

To print the pipeline editor

1. In the Service Browser, select the flow service for which you want to print the pipeline editor.
2. In the Flow Editor, select the INVOKE or MAP step for which you want to print the pipeline editor.
3. Click anywhere on the Pipeline tab.
4. Scroll or resize the Pipeline tab to display the portion of the pipeline editor you want to view as HTML.
5. From the File menu, click View as HTML or click .
   
   Developer creates an HTML page and displays it in your default browser.
6. If you want to print the pipeline editor, use your browser's print command.

Basic Mapping Tasks

Basic mapping tasks are the tasks you perform to manage the pipeline contents and the values of variables in the pipeline. In the Pipeline Editor, you can perform the following basic mapping tasks:

- **Map variables to each other.** You can copy the value of a variable in one service or document format to a variable in another service or document format.

- **Assign values to variables.** You can hard code variable values or assign a default value to variables.

- **Drop variables from the pipeline.** You can remove pipeline variables that are not used by subsequent services in a flow.

- **Add variables to the pipeline.** You can add variables that were not declared as input or output parameters of the flow service. You can also add input and output variables for services that the flow service invokes (internally invoked services).
The following table identifies the sections that describe the basic mapping tasks.

<table>
<thead>
<tr>
<th>For more information about...</th>
<th>See page...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping variables</td>
<td>155</td>
</tr>
<tr>
<td>Mapping variables of different data types</td>
<td>163</td>
</tr>
<tr>
<td>Mapping to and from array variables</td>
<td>165</td>
</tr>
<tr>
<td>Deleting links between variables</td>
<td>169</td>
</tr>
<tr>
<td>Applying conditions to maps between variables</td>
<td>169</td>
</tr>
<tr>
<td>Assigning values to pipeline variables</td>
<td>172</td>
</tr>
<tr>
<td>Dropping variables from the pipeline</td>
<td>175</td>
</tr>
<tr>
<td>Adding variables to the pipeline</td>
<td>176</td>
</tr>
</tbody>
</table>

**Mapping Variables**

When you want to copy the value of a variable in a service or document format to another variable, you map the variables. The Pipeline Editor connects service and pipeline variables with a line called a *link*. The process of mapping variables is also called creating a *link*.

Within a flow, Developer implicitly maps variables whose names are the same and whose data types are compatible. For example, the service in the following flow takes a variable called `ActNumber`. Because a variable by this name already exists in Pipeline In, it is automatically mapped to the `ActNumber` variable in Service In. The Pipeline Editor connects implicitly mapped variables with a gray link.
Implicit maps between pipeline and service variables

Pipeline variables are automatically mapped to service variables of the same name.

Important! The Pipeline Editor does not display implicit mapping lines for a MAP step.

In cases where the services in a flow do not use the same names for a piece of information, use the Pipeline Editor to explicitly map the variables to each other. Explicit mapping is how you accomplish name and structure transformations required in a flow. The Pipeline Editor connects explicitly mapped variables with a solid black line.

On the input side of the Pipeline Editor, use the Map modifier to map a variable from the pipeline to the service. In the following example, the service expects a value called OrderTotal, which is equivalent to the pipeline variable BuyersTotal—they are simply different names for the same data. To use the value of BuyersTotal as the value for OrderTotal, you “map” the pipeline variable to the service using the Map modifier.

At run time, the server will copy the value from the source variable (BuyersTotal) to the target variable (OrderTotal) before executing the service.
Mapping the pipeline to service input

All the output variables that a service produces are automatically placed in the pipeline. Just as you can map variables from the Pipeline In stage to a service’s input variables, you can map the output from a service to a different variable in Pipeline Out.

In the following example, a variable called TransactionNumber is mapped to the element Num in a record called TransRecord. At run time, the server will copy the value of TransactionNumber to Num, and both TransactionNumber and Num will be available to subsequent services in the flow.
When you map variables in the pipeline, keep the following points in mind:

- The variable that you are mapping from is the source. For example, when you map from a variable in Pipeline In to one in Service In, the Pipeline In variable is the source. When you map from a variable in Service Out to one in Pipeline Out, the Service Out variable is the source.

- The variable you are mapping to is the target. For example, when you map from a variable in Pipeline In to one in Service In, the Service In variable is the target. When you map from a variable in Service Out to one in Pipeline Out, the Pipeline Out variable is the target.

- A Service In variable can be the target of more than one Map modifier only if you use array indexing or if you place conditions on the maps to the variable.

- By mapping variables to each other, you are copying data from the source variable to the target variable. (Records, however, are copied by reference. For more information, see “What Happens When SAP BC Server Executes a Map Between Variables at Run Time?” on page 160.)
After you map to a target variable, you cannot create another map to the target variable—a target variable can be connected to only one source variable. (Two exceptions to this rule involve array variables and conditional maps. For more information about mapping array variables, see “Mapping to and from Array Variables” on page 165. For more information about placing conditions on maps between variables, see “Applying Conditions to Maps Between Variables” on page 169.

You cannot create a map to a variable if you already used the Set Value modifier to assign a value to a variable.

After a Map modifier is executed, both the source and target variables exist in the pipeline. The target variable does not replace the source variable.

To map one variable to another

1 In the Flow Editor, select the INVOKE or MAP step containing the variables you want to map.

2 Click the Pipeline tab.

3 If you want to map a variable from Pipeline In to Service In, do the following:
   1 In Pipeline In, click the pipeline variable you want to use as the source variable.
   2 In Service In, click the input variable you want to use as the target variable.
   3 Click ☰ on the toolbar.

4 If you want to map a variable from Service Out to Pipeline Out, do the following:
   1 In Service Out, click the output variable you want to use as the source variable.
   2 In Pipeline Out, click the pipeline variable you want to use as the target variable.
   3 Click ☰ on the toolbar.

Notes:
— If the variable types are incompatible and cannot be mapped to one another, the Pipeline Editor displays a message stating that the operation is not allowed.
— If you mapped to or from an array variable, you need to specify which element in the array you are mapping to or from. For more information about array mapping, see “Mapping to and from Array Variables” on page 165.
— If you want to place a condition on the execution of the map, see “Applying Conditions to Maps Between Variables” on page 169.

Tip! You can also use your mouse to map variables to one another. To do this, select the source variable and drag your mouse to the appropriate target variable.
What Happens When SAP BC Server Executes a Map Between Variables at Run Time?

When executing a map between variables at run time, SAP BC Server does one of the following:

- Copies the value from the source variable to the target variable. For example, when you map from a source String variable to a target String variable, SAP BC Server copies the value of the source String to the target String. This is called “copying by value.”

- Creates a reference to the source variable and uses the reference as the value of the target variable. For example, when executing a map between a source Record variable and a target Record variable, SAP BC Server creates a reference to the source Record value and uses the reference as the value of the target Record variable. This is called “copying by reference.”

SAP BC Server copies by value when the source or target variable is a String. The variables can be children of Records or Record Lists—as long as the source or target variable is a String, SAP BC Server copies by value. (An exception to this is that when executing a map from a String to an Object, SAP BC Server copies by reference.)

When executing maps between all other types of variables, SAP BC Server copies by reference. Copying by reference significantly reduces the memory and time required for executing a map at run time.

When a value is copied by reference, any changes you make to the value of the source variable in subsequent flow steps affect the target variable. This is because the value of the source variable is the value of the target variable—not a copy of the source variable value.

Consequently, any changes you make to the value of the source variable affect the target variable. If, in a later flow step, you used the Set Value modifier to assign a value to the source variable, you would be changing the value of the target variable as well—the target variable references the value of the source variable.

The following images show a series of MAP steps in a flow service. In this example, the value of the source variable is changed after the map to the target variable executes. This action changes the value of the target variable as well.
Step 1: The value of String1 is set to “original value”

The value of String1 is set to "original value".

Step 2: R1 is mapped to R2

R1 is mapped to R2. After the map executes, the value of R2 is a reference to the contents of R1.

Step 3: The value of String1 is changed to “modified” after the map executes

The value of String1 is changed to "modified". This action changes the value of the string in R2 as well.
When the above flow service executes, it returns the following results.

### Results of flow service

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1 String1</td>
<td>modified</td>
</tr>
<tr>
<td>R2 String1</td>
<td>modified</td>
</tr>
</tbody>
</table>

In Step 3, the value of the String1 in R1 was set to “modified.” However, the value of String1 in R2 changed also. This is because in step 2 of the flow service, the value of R1 was copied to R2 by reference. Changes to the value of R1 in later flow steps also change the value of R2.

To prevent the value of the target variable from being overwritten by changes to the value of the source value in subsequent steps in the flow service, you can do one of the following:

- When working with Record variables, map each child of the Record variable individually. This method can be time consuming and might significantly increase the memory and time required to run the service. However, this might be the best approach if the target record needs only a few values from the source record.

- After you map the source variable to a target variable, use the Drop modifier to drop the source variable. Only the target variable will have the reference to the data. This method ensures that the value of the target variable will not be overwritten in a subsequent step, but does not increase the memory and time required to execute the service.

- Create a service that performs a copy by value. Insert this service (as an INVOKE step or as a transformer) and map the variables to the service instead of mapping them to each other. (In the case of Records, you could create a Java service that clones the IData object underlying the Record.) In situations where you map one Record to another, using a “cloning” service would require less time than mapping a record field by field.
Mapping to Record Variables

When working with Records in the pipeline, you can map a source variable to the Record variable or to the children of the Record variable. Keep the following points in mind when mapping to Record or Record List variables:

- A Record (or a Record List) and its children cannot both be targets. After a Record or Record List is the target of a Map modifier, its children cannot be the targets of Map modifiers.

- After the child variable of a Record or Record List is the target of a Map modifier, the parent Record or Record List cannot be a target of a Map modifier.

- If you map from a Record variable to another Record variable, the structure of the source Record variable overwrites the structure of the target Record variable.

Mapping Variables of Different Data Types

The Pipeline Editor allows you to map different, but compatible, data types to one another. For example, you could map a String value called AccountNumber to a String List called Accounts. At run time, the server automatically performs the structural transformation necessary to map the data in AccountNumber to Accounts. (In this case, the transformation will result in a single-element String List.) By mapping different data types to one another, you can perform structural transformations.

If you map variables of different data types, keep the following points in mind:

- Not all data types can be mapped to one another. You cannot map a Record to a String, for instance. If two data types are incompatible, the Pipeline Editor will not allow you to connect them with the Map modifier.

- You can only map a variable to another variable of the same primitive type. The primitive type refers to the data type of the variable when all dimensionality is removed. For example, the primitive type for a String List or a String Table would be String. Two exceptions to this rule are: any variable can be mapped to an Object or an Object List variable, and an Object can be mapped to any data type. (If there is a type mismatch between the Object or Object List and the other variable at run time, Developer does not perform the map.)

- When you map between scalar and array variables, you can specify which element of the array variable want to map to or from. Scalar variables are those that hold a single value, such as String, Record, and Object. Array variables are those that hold multiple values, such as String List, String Table, Record List, and Object List. For example, you can map a String to the second element of a String List. Alternatively, you can map the second element in a String List to a String. For more information about mapping array variables, see “Mapping to and from Array Variables” on page 165.
When you map between scalar and array variables and you do not specify which element in the array variable that you want to map to or from, Developer uses the default rules in the Pipeline Editor to determine the value of the target variable. For more information about the default behavior for mapping array variables, see “Default Pipeline Rules for Mapping to and from Array Variables” on page 580.

The following table identifies which data types you can map to each other.

<table>
<thead>
<tr>
<th>You can map this data type...</th>
<th>To these data types...</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>T</td>
</tr>
<tr>
<td>String List</td>
<td>T</td>
</tr>
<tr>
<td>String Table</td>
<td>T</td>
</tr>
<tr>
<td>Record</td>
<td>T</td>
</tr>
<tr>
<td>Record List</td>
<td>T</td>
</tr>
<tr>
<td>Record Reference</td>
<td>T</td>
</tr>
<tr>
<td>Record Reference List</td>
<td>T</td>
</tr>
<tr>
<td>Object</td>
<td>*</td>
</tr>
<tr>
<td>Object List</td>
<td>*</td>
</tr>
</tbody>
</table>

Examples of Structural Transformations in the Pipeline Editor

The structural transformations you can perform by mapping variables in the Pipeline Editor can be more complex than transforming a String to a String List. For example, you can combine two String Lists into one Record List through mapping. The following section explains a common structural transformation that you can complete via mapping in the pipeline.

Converting a String List to a Record List

You can convert a String List to a Record List using the Pipeline Editor. In the following diagram, aList is the String List you want to convert to a Record List. The variable recList is the Record List to which you want to copy the values contained in the String List. recList has a String child aString. To convert the String List to a Record List, map aList to aString.
Converting a String List to a Record List

Two String Lists can be combined into one Record List through pipeline mapping. For example, if in the above scenario, you also had a String List variable named bList, and recList had two String children, aString and bString, you could combine the two String Lists by mapping aList to aString and bList to bString.

Converting Two String Lists to a Record List

Tip! You can also convert a String List to a Record List by invoking the built-in service pub.list:stringListToRecordList. You can insert the service as an INVOKE step or as a transformer. For more information about transformers, see “What Are Transformers?” on page 177. For more information about built-in services, see the SAP BC Built-In Services Guide.

Mapping to and from Array Variables

When you map to or from an array variable (String List, String Table, Record List, or Object List), you can specify which element in the array you want to map to or from. After you map the variables, you specify the index that represents the position of the element in the array.
For String Lists and Object Lists, you can specify the index for the element in the list that you want to map. For example, you can map the third element in a String List to a String.

For String Tables, you can specify the row and column indexes for the element that you want to map. For example, you can map the value of the element in the second row and third column of a String Table to a String.

For Record Lists, you can specify the index for the record in the Record List that you want to map. For example, you can map the second record in a Record List to a Record variable.

For a variable in a Record List, you can specify the index for the record in the Record list that contains the value that you want to map. For example, if your source variable (a String named ItemNumber) is the child of a Record List named POItems, you can map the value of ItemNumber from the second POItems record to a String variable.

In the following diagram, the AccountNumber String variable is mapped to the Accounts String List variable.

You can specify an index value when mapping to an array variable

![Diagram showing mapping and indexing]

To specify the index for the element in the Accounts variable to which you want to map the AccountNumber value, select the link between the variables and select Edit > Properties. Then, use the Indexing tab on the Link Properties dialog box to specify the index. If the source or target variable is an array, Developer displays a text box next to the variable (in this case, Accounts). For example, if you want to map the value of the AccountNumber variable to the second element in the Accounts String List, type 1 in the field next to Accounts. (Index numbering in arrays begins at 0.)
If the source or target variable is not an array, Developer displays the words “Field not indexable” next to the variable name.

Indexing tab

Guidelines for Mapping to and from Array Variables

When you are mapping to or from an array variable, keep the following points in mind:

- To map to or from an element in an array variable, you need to know the index for the element's position in the array. Remember, array index numbering begins at 0 (the first element in the array has an index of 0, the second element has an index of 1).

- If you map to an array variable and specify an index that does not exist, Developer increases the length of the array variable to include the specified array index. For example, if a String List has length 3, but you specify that you want to map a String to array index 5 in the String List, Developer increases the length of the String List to 6.

- Each element in an array can be the source or target of a Map modifier—that is, each element in the array can be the start or end of a link. For example, if a source String List variable contains three elements, you can map each of the three elements to a target variable.
If the source and target variables are arrays, you can specify an index for each variable. For example, you can map the third element in a source String List to the fifth element in target String List.

If you do not specify an array index for an element when mapping to or from arrays, the default behavior of the Pipeline Editor will be used. For information about the default behavior of the Pipeline Editor, see “Default Pipeline Rules for Mapping to and from Array Variables” on page 580.

The following procedure explains how to map to or from an array variable.

To map to or from an array variable

1. Map the variables to each other using the procedure described in “To map one variable to another” on page 159.
2. Right-click the link (line) that connects the variables, and select Properties. (You can also double-click the link to display the Link Properties dialog box.)
3. In the Link Properties dialog box, click the Indexing tab.
4. If the source variable is an array variable, under Source, next to the source variable name, type the index that contains the value you want to map.
5. If the target variable is an array variable, under Destination, next to the Destination variable name, type the index to which you want to map the source value.
6. Click OK.

**Note:** If you are mapping to or from a String Table, you need to specify an index value for the row and column.

**Note:** When you map a Record or Record List variable to another Record or Record List variable, the structure of the source variable determines the structure of the target variable. For more information, see “Mapping to Record Variables” on page 163.
Deleting Links Between Variables

Use the following procedure to delete the link created by mapping two variables to each other. When you delete the link, the variables are no longer mapped. Developer also deletes any properties you applied to the link.

To delete a link between variables

- In the Pipeline Editor, select the link that you want to delete, and click delete, or right-click and select Delete from the shortcut menu.

Tip! You can also delete a link by selecting it and then pressing the DELETE key.

Applying Conditions to Maps Between Variables

You can place conditions on the maps (links) you draw between variables. At run time, SAP BC Server evaluates the condition and executes the map (copies the value) only if the condition evaluates to true.

A condition consists of one or more expressions that you write using the syntax that Developer provides. An expression can check for the existence of a variable in the pipeline, check for the value of a variable, or compare a variable to another variable. For example, in the following service, you might want to map the BuyersTotal variable in Pipeline In to the OrderTotal variable in Service In only if the BuyersTotal has a value that is not null. After you connect the two variables with the Map modifier, you would edit the properties and add the condition that needs to be true.
A blue link indicates that a condition is applied to the link connecting the variables.

The Pipeline Editor uses blue links to indicate that properties—conditions or index values for arrays—have been applied to the links (map) between variables.

**Note:** You cannot add conditions to the links between implicitly mapped variables.

**Mapping Multiple Source Variables to a Target Variable**

By applying conditions to the maps between variables, you can map more than one source variable to the same target variable. When you draw more than one map to the same target variable, at most, only *one* of the conditions you apply to the links can be true at run time—the conditions need to be mutually exclusive.

At run time, SAP BC Server executes all conditional maps whose conditions evaluate to true. If more than one conditional map to the same target variable evaluates to true, the value of the target variable will be the result of whichever map executes last. Because the order in which maps are executed at run time is not guaranteed, the final value of the target variable may vary.

**Tip!** If the conditions for maps to the same target variable are not mutually exclusive, consider using a flow service containing a BRANCH step instead. In BRANCH steps, child steps are evaluated in a top to bottom sequence. SAP BC Server executes the first child step that evaluates to true and skips the remaining child steps. For more information about the BRANCH step, see “The BRANCH Step” on page 110.
To apply a condition to the link between variables

1. Map the variables to each other using the procedure described in “To map one variable to another” on page 159.
2. Right-click the link (black line) that connects the variables, and select Properties.
3. In the Link Properties dialog box, click the General tab.

General tab

4. Select the Copy only if check box.
5. In the text box on the General tab, type the condition you want to place on the link. Click OK.

For information about the syntax used in conditions, see Appendix E, “Conditional Expressions” on page 593

**Important!** When drawing more than one map to the same target variable, make sure that the conditions assigned to each map are mutually exclusive.

**Note:** You can temporarily disable the condition placed on a map. For more information, see “Disabling a Condition Placed on a Map Between Variables” on page 273.
Assigning Values to Pipeline Variables

The Set Value modifier allows you to assign values to variables in Service In or Pipeline Out. You use it to explicitly “hard code” a specific value in a variable. You can also use it to assign a default value to a variable.

By attaching a Set Value modifier to a variable, you instruct the server to write a specific value to that variable at run time. This action occurs just before the selected service is executed (if you attach the modifier to a variable in Service In) or immediately after the selected service is executed (if you attach the modifier to a variable in Pipeline Out).

Hardcoding the value of a variable

To view (or change) the value that is assigned to the Set Value modifier, double-click the icon next to the variable's name to open the Input For dialog box.

Input For dialog box
Assigning a Default Value to a Variable

One common use of the Set Value modifier is to specify a default value for a variable—a value that is only assigned if the variable is null at runtime. To use the Set Value modifier in this way, disable the Overwrite Pipeline Value check box in the Input For dialog box. This instructs the server to use the specified value only when the selected variable is null.

Note: If a variable to which you assigned a default value is implicitly mapped to another variable in the pipeline, the Pipeline Editor displays a gray link connecting the variables beneath the icon.

Initializing Variables in a Flow Service

You can use the Set Value modifier with the MAP step to hard code an initial set of input values in a flow service. To use it in this way, insert the MAP at the beginning of your flow, and then use the Set Value modifier to assign values to the appropriate variables in Pipeline Out.

Referencing Other Variables

In addition to assigning a literal value to a variable, the Set Value modifier lets you assign the value of another variable to a variable. (You might do this if you wanted to derive the default value from a variable in the pipeline at runtime, for example.)

To specify a variable name with the Set Value modifier, enclose the name of that variable between % symbols and then enable the Perform Variable Substitution option. This option instructs the server to interpret your value as a variable reference rather than a literal value.

You can also format string values by specifying one or more pipeline variables in conjunction with a literal value. For example, if you specified (%areaCode%) %Phone%, the resulting string would be formatted to include the parentheses and space. If you specified %firstName% @initial%. %lastName%, the period and spacing would be included in the value.
CHAPTER 7  Mapping Data in a Flow Service

To assign a value to a variable in the Pipeline

1. In the Flow Editor, select the INVOKE or MAP step containing the variable you want to alter.
2. Click the Pipeline tab.
3. Select the variable to which you want to assign a value.

**Note:** You can only assign values to variables that are in Service In or Pipeline Out.

**Note:** If the Service In or Pipeline Out variable is already the target for a Map modifier, you cannot use the Set Value modifier to assign a value to the variable.

4. Click on the toolbar.
5. In the Input For dialog box, specify the value you want to assign to this variable.
   - If you want to assign a literal value to the variable, type that value.
   - If you want to derive the value from a variable in the pipeline, type the name of that variable enclosed in % symbols (e.g., %Phone%). Then, select the Perform Variable Substitution check box.
6. If you want the server to use the specified value only if the variable does not contain a value at run time, clear the Overwrite Pipeline Value check box. (If you select this check box, the server will always apply the specified value.)

**Note:** You can mix literal values and pipeline variables

**Copying Set Values Between Variables**

You can copy the set value assigned to a variable to other variables of the same data type in Service In or Pipeline Out. When you copy set values from one pipeline variable to another, keep the following points in mind:

- You can only copy and paste set values between variables of the same data type. (For example, you can only copy the set value assigned to a String variable to another String variable.)
- You can only copy and paste set values between variables if the target variable has the same structure as the source variable or has no defined structure. For example, you can copy the set value of a String List variable with length 3 to another String List variable only if the target String List also has length 3 or has an undefined length (no defined structure).
If you are copying a set value between Record variables, the source Record variable and the target Record variable must have the same structure or the target Record variable must have no structure defined. (For example, if the source Record variable is a record named `AddressInfo` and has three String variables named `City`, `State`, and `Zip` as children, the target Record variable must have three String variables named `City`, `State`, and `Zip` as children.)

**To copy a set value**

1. In the Flow Editor, select the INVOKE or MAP step containing the variable with the value you want to copy and paste.
2. Click the **Pipeline** tab.
3. Select the **variable** you want to copy.
4. Right-click and select **Copy**.
5. Select the variable or variables to which you want to assign the copied value, right-click and select **Paste**.

**Note:** You can only copy and paste values for variables that are in **Service In** or **Pipeline Out**.

**Note:** You can only paste the set value if the target variable is the same data type as the source variable and if the target variable has either an identical structure to the source variable or has no defined structure.

### Dropping Variables from the Pipeline

The **Drop** pipeline modifier allows you to remove a variable from **Pipeline In** or **Pipeline Out**. You can use it to eliminate pipeline variables that are not used by subsequent services in a flow. Dropping unneeded variables reduces the size of the pipeline at run time and reduces the length and complexity of the **Pipeline In** and **Pipeline Out** displays (this can make the Pipeline Editor much easier to use when you are working with a complex flow).

**Important!** Once you drop a variable from the pipeline, it is no longer available to subsequent services in the flow. Do not use the **Drop** modifier unless you are sure the variable is not used by services invoked after the point where you drop it.

At run time, the server removes a dropped variable from the pipeline just before it executes the selected service (if you attach the **Drop** modifier to a variable in **Pipeline In**) or immediately after it executes the selected service (if you attach the **Drop** modifier to a variable in **Pipeline Out**).
If you drop a mapped variable from Pipeline In, the server executes the Map modifier before it drops the variable. The server does not map a null value to the destination variable.

To drop a variable from the Pipeline

1. In the Flow Editor, select the INVOKE or MAP step whose pipeline variables you want to drop.
2. Click the Pipeline tab.
3. Select the variable that you want to drop.
4. Click on the toolbar.

Note: You can only drop variables from Pipeline In and Pipeline Out. In a MAP step, you can only drop variables from Pipeline In.

Adding Variables with the Pipeline Editor

The Pipeline Editor allows you to add variables that were not declared as input or output parameters for the flow service itself or any of its constituent services. You can use it to add variables that were omitted from a service’s input or output parameters or create temporary variables for use within the flow. (For example, you might attach a variable to each of the children in a BRANCH step to mark the path taken by the service at run time.) Variables that you create with the Pipeline Editor can be used just like any declared variable in the flow.

Important! If you create a new variable in a flow, you must immediately do one of the following:

– Map a variable to it
– Assign a value to it
– Drop it

If you do not take one of these steps, the Pipeline Editor automatically clears it from the map the next time it refreshes the Pipeline tab.

Note: You might want to drop a variable immediately after adding it if a service produces a variable that is not declared in the service input or output parameters. The variable will not appear in the Pipeline Editor if it is not an input or output parameter. By adding and then immediately dropping the variable, you can delete the variable if it does exist in the pipeline.
To add a variable to the Pipeline

1. In the Flow Editor, select the INVOKE or MAP step that represents the stage of the pipeline at which you want to add a new variable.

2. Click the Pipeline tab.

3. Select the point where you want to add the new variable.

4. Click \[\text{variable type}\] and select the type of variable that you want to create.

5. Type the name of the variable and press ENTER.

6. If the variable is a Record or a Record List, repeat steps 4 and 5 to define its member variables. Then use \[\text{indent symbol}\] to indent each member variable beneath the Record or Record List variable.

7. Assign one of the pipeline modifiers to the new variable (Map, Drop, or Set Value). (If you do not assign a modifier to the variable, the Pipeline Editor considers it extraneous to the flow and automatically clears the variable when it refreshes the Pipeline tab.)

Working with Transformers

By mapping variables to each other in the Pipeline Editor, you can accomplish name transformations and structural transformations. However, to perform value transformations you need to execute some code or logic—that is, you need to invoke a service. Developer provides two ways for you to invoke services—you can insert INVOKE steps or you can insert transformers into the Pipeline Editor.

What Are Transformers?

Transformers are the services you use to accomplish value transformations in the Pipeline Editor. You can only insert a transformer into a MAP step. You can use any service as a transformer. This includes any Java, C or flow service that you create and any built-in services in WmPublic, such as the pub.date.currentDate and the pub.string.concat services. By using transformers, you can invoke multiple services (and perform multiple value transformations) in a single flow step.

Note: Services that you insert using the INVOKE step might also perform value transformations. However, only transformers can accomplish multiple value transformations in a single flow step.
You can think of transformers as a series of INVOKE steps embedded in a MAP step. And like INVOKE steps, when you insert a transformer, you need to map variables between the pipeline to the transformer (create links between pipeline variables and the transformer). You can also set properties for the transformer and validate the input and/or output of the transformer. Because transformers are contained within a MAP step, they do not appear as a separate flow step in the Flow Editor.

Transformers are well suited for use when mapping data from one document format to another. When you map data between formats, you usually need to perform several name, structure, and value transformations. By using transformers, the flow service in which you map data between formats could potentially consist of a single MAP step in where transformers and links between variables handle all of the data transformations. In this way, you could see your entire document-to-document mapping in a single view.

Tip! You can create a flow service that uses transformers to convert data between document formats (such as an IDOC to an XML document or RosettaNet PIP to a proprietary format). You could then invoke this service in other flow services each time you need to convert between the specific document formats before you begin processing data.

MAP Step with transformers

Note: In a MAP step, the Pipeline Editor only displays the links between pipeline variables and transformers. The Pipeline Editor does not display any implicit mapping for a MAP step.
Using Built-in Services as Transformers

Any service in the Service Browser can be used as a transformer. SAP provides several built-in services specifically designed to translate values between formats. These services can be found in the following folders in the WmPublic package:

<table>
<thead>
<tr>
<th>This folder...</th>
<th>Contains services to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>pub.date</td>
<td>Transform time and date information from one format to another.</td>
</tr>
<tr>
<td>pub.list</td>
<td>Transform a String List to a Record List and append items to a Record List or a String List.</td>
</tr>
<tr>
<td>pub.math</td>
<td>Perform simple arithmetic calculations (add, subtract, multiply, and divide) on integers and decimals contained in string variables.</td>
</tr>
<tr>
<td>pub.record</td>
<td>Transform records to and from record lists and SAP BC XML values.</td>
</tr>
<tr>
<td>pub.string</td>
<td>Transform string values in various ways (e.g., pad, substring, concat, replace through a lookup table).</td>
</tr>
</tbody>
</table>

For more information about built-in services, see the SAP BC Built-In Services Guide.

Inserting a Transformer into a MAP Step

When you insert a transformer, you are essentially inserting an INVOKE step into a MAP step. When inserting transformers, keep the following items in mind:

- Transformers can only be inserted in a MAP step.
- Any service can be used as a transformer, including flow services, C services, and Java services.
- The transformers you insert into a MAP step operate on the same set of pipeline data.
- The output of one transformer cannot be used as the input of another transformer in the same MAP step.
- Transformers in a MAP step are independent of each other and do not execute in a specific order. When inserting transformers, assume that SAP BC Server concurrently executes the transformers at run time.

To insert a transformer

1. In the Flow Editor, select the MAP step in which you want to insert a transformer.
2. Click the Pipeline tab.
3 Click \[\text{Edit}\] on the Pipeline tab toolbar, and select the service you want to invoke. If the service you want to insert does not appear in the list, select Browse to select the service from the Service Browser. The transformer appears under Transformers on the Pipeline tab.

4 To set the properties for the transformer, right-click the transformer and select Properties. Then, complete the following fields in the Transformer Properties dialog box:

   For this property... Specify...
   service The service that will be invoked at run time. When you insert a transformer, Developer automatically assigns the name of that service to the service property. If you want to change the service that is invoked by a transformer, specify the service’s fully qualified name in the footerName:serviceName format.
   validate-in Whether or not you want to validate the input to the transformer. Select $default to validate the input of the transformer. Select $none if you do not want to validate the input of the transformer.
   validate-out Whether or not you want to validate the output of the transformer. Select $default to validate the output of the transformer. Select $none if you do not want to validate the output of the transformer.

5 Click OK.

For information about debugging transformers, see “Debugging Transformers” on page 187.

Tip! When you zoom in on the transformer, you can see the Service In variables and the Service Out variables and all of the explicit maps between the transformer and the pipeline. You might find it easier to map transformer variables when you are zoomed in on the transformer.

Note: In SAP BC version 3.x, you could set an as-user property for a transformer. In SAP BC Server version 4.0, this property was removed. For more information see “The As-User Property in SAP BC Developer 3.x” on page 97.
Mapping Variables to a Transformer

When you map data to and from a transformer, you create links between the pipeline variables and the transformer. Keep the following points in mind when you map variables to a transformer:

- Developer does not implicitly map pipeline variables to the input or output variables of a transformer. Even if the pipeline variables have the same name and data type as the transformer variables, no implicit mapping occurs. You need to explicitly map pipeline variables to the input and output variables of a transformer.

- Output for a transformer is not automatically added to the pipeline. If you want the output of a transformer to appear in the pipeline, you need to explicitly map the output variable to a Pipeline Out variable. If you do not map the output variable to a Pipeline Out variable, the output variable does not appear in the pipeline.

- If you do not map any output variables or the transformer does not have any declared output variables, the transformer service will not run.

- The transformers you insert into a single MAP step act on the same set of pipeline data.

To provide the cleanest and simplest view when working with transformers, the Pipeline Editor only displays one link between the transformer and a Pipeline In variable and one link between the transformer and a Pipeline Out variable. (The Pipeline Editor displays the links between the transformer and the highest positioned Pipeline In variable and Pipeline Out variable to which the transformer is mapped.)

To map variables to a transformer

1. To map a variable from Pipeline In to the transformer, do the following:

   1. In Pipeline In, select the variable you want to use as input to the transformer and drag your mouse to the transformer.

   2. In the Map To list, select the transformer variable to which you want to map the Pipeline In variable.

      Once you map input variable for a transformer to a Pipeline In variable, Developer displays the phrase “has already been chosen” next to the transformer variable in the Map To list.

   3. Repeat steps 1 and 2 for each transformer input variable you want to map to a pipeline variable.

Note: You can use the Set Value modifier to assign a value to a transformer input variable. To use the Set Value modifier with a transformer, you need to expand the transformer. To expand, double-click the transformer name, click Expand on the Compose menu, or click to next to the transformer.
2. To map an output variable from the transformer to **Pipeline Out**, do the following:

1. Select the transformer and drag your mouse to the variable in **Pipeline Out** to which you want to map the transformer.

2. In the **Map From** list, select the transformer variable that you want to map to the selected **Pipeline Out** variable.

3. Repeat steps 1 and 2 for each output variable produced by the transformer.

You can map an output variable for a transformer to more than one **Pipeline Out** variable.

---

**Important!** Developer does not automatically add the output of a transformer to the pipeline. If you want the output of a transformer to appear in the pipeline after the transformer executes, you need to explicitly map the output variable to a variable in **Pipeline Out**.

---

**Important!** If you do not map any output variables or the transformer does not have any output variables, the transformer will not execute.

---

**Transformer Movement**

When you map to and from a selected transformer, it moves up and down in the **Transformers** column. This movement or “jumping” is by design to help minimize the distance between the transformer and the variable you are mapping it to.

Transformers exhibit the following behavior or movement:

- When a transformer is selected and you select a variable in **Pipeline In** or **Pipeline Out**, the transformer “jumps” or moves up or down in the **Transformers** column so that it is directly across from the selected pipeline variable.

- When you finish mapping a transformer and it is no longer selected, the Pipeline Editor “anchors” or aligns the transformer next to the highest **Pipeline Out** variable it is mapped to.

To stop the transformer from jumping, click the transformer again or click in the empty areas of the Pipeline Editor.

---

**Note:** To expand your view of the Pipeline Editor, maximize the **Pipeline** tab. Click anywhere on the **Pipeline** tab, and click on the toolbar. The **Pipeline** tab expands to be the size of the Developer window.
Transformers and Array Variables

When mapping to and from transformers, differences in dimensionality for the source and target variables may cause an exception. If the dimensionality of the target variable is greater than the dimensionality of the source variable, an exception will not be thrown. However, if the dimension of the source variable is greater than the dimension of the target variable, an exception will occur.

What Is Dimensionality?

Dimensionality refers to the number of arrays to which a variable belongs. For example, the dimensionality of a single String is 0, that of a single String List or Record List is 1, and that of a single String Table is 2. A String that is a child of a Record List has a dimensionality of 1. A String List that is a child of a Record List has a dimensionality of 2.

Example

In the following example, the unitPrice variable cannot be mapped to num1 because the unitPrice variable has a dimensionality of 1 (string (0) + Record List (1) = 1) and num1 has a dimension of 0.

*unitPrice* cannot be mapped to *num1* because of dimensionality differences
CHAPTER 7 Mapping Data in a Flow Service

Solution

To solve this, you can either:

- Change the service invoked by the transformer to accept arrays as data, or
- Create a flow service in which a LOOP step loops over the array variable. Then, (in the same flow service) invoke the service you originally wanted to use as a transformer, and make that INVOKE step a child of the LOOP. Finally, insert the resulting flow service as a transformer in the MAP.

Of the two options, changing the service to accept arrays as data results in faster execution of flow services.

Validating Input and Output for Transformers

As with any service you insert using an INVOKE, you can validate the inputs and outputs of the transformer service before and/or after it executes. To indicate that you want to validate a transformer’s inputs and outputs, you change the properties of the transformer. You do not have to use validation for all of the transformers you insert into a MAP step.

When the server validates a transformer’s inputs and outputs at run time, it validates the transformer against the input and output parameters of the invoked service. To view the input and output parameters of the invoked service, select the service in the Service Browser, and click the Input/Output tab. The variables on the input and output sides of the tab represent the declared parameters for the service. To view the constraints placed on a variable, right-click the variable, select Properties, and then click the Constraints tab.

Note: If the Validate in and/or Validate out check boxes are selected on the Input/Output tab of the invoked service, then the input and/or output for the service will automatically be validated every time the service is executed. If you set up validation via the properties for a transformer when it is already set up for validation via the Input/Output tab, then you are performing validation twice. This can slow down the execution of a transformer and, ultimately, the flow service.

To validate the input and output of a transformer

1. In the Flow Editor, select the MAP step containing the transformer you want to validate.
2. Click the Pipeline tab.
3. Under Transformers, select the transformer for which you want to validate input or output. Right-click the transformer, and select Properties.
4. In the Transformer Properties dialog box, in the validate-in list, select $default if you want to validate the input to the transformer against the input parameters of the invoked service.
5 In the validate-out list, select $default if you want to validate the output of the transformer against the output parameters of the invoked service.

6 Click OK.

Copying Transformers

You may want to use the same transformer more than once in a MAP step. For example, you might want to convert all the dates in a purchase order to the same format. Instead of using the button to locate and select the service, you can copy and paste the transformer service.

You can also copy transformers between MAP steps in the same flow or MAP steps in different flow services.

**Important!** Copying a transformer does not copy the links between transformer variables and pipeline variables or any values you might have assigned to transformer variables using the Set Value modifier.

To copy a transformer

1 In the Flow Editor, select the MAP step containing the transformer service you want to copy.

2 Click the Pipeline tab.

3 Under Transformers, select the transformer service you want to copy. Right-click the transformer, and select Copy.

4 To paste the transformer, click anywhere under Transformers. Right-click and select Paste.

5 Map the input and output variables of the transformer using the procedures described in “Mapping Variables to a Transformer” on page 181.

Expanding Transformers

You might find it easier to map transformers when you expand the transformer. When you expand a transformer, you can see the Service In and the Service Out variables for the transformer and all of the maps between the pipeline and the transformer variables.
When you expand a transformer, you can only perform actions for that transformer, e.g., you can only map variables or set properties for the expanded transformer. Other transformers and mappings to other transformers remain hidden until you collapse the transformer.

**Note:** If you expand a transformer, you can use the **Set Value** modifier to assign a value to a variable in **Service In**.

### To expand a transformer

- On the **Compose** menu, click **Expand**.
  - OR –
- Double-click the transformer you want to expand.
  - OR –
- Click ⬤ next to the transformer you want to expand.

To return to a normal view of the **Pipeline** tab, click – next to the transformer name, or on the **Compose** menu, click **Collapse**.

**Note:** If SAP BC Server displays a message stating that the transformer cannot be found, then the service invoked by the transformer has been renamed, moved, or deleted. You need to use the transformer properties to rename the transformer. See below for more information.
Renaming Transformers

If SAP BC Server displays the message “Transformer not found” when you try to expand a transformer or when you point the mouse to the transformer, then the service referenced by the transformer has been renamed, moved, or deleted. You need to change the service property of the transformer so that the transformer points to the moved, or renamed service.

If the service referenced by the transformer has been deleted, you may want to delete the transformer.

Tip! You can enable safeguards so that you do not inadvertently affect or break other services when you move, rename, or delete a service. For more information, see “Renaming and Deleting Elements” on page 37.

To rename a transformer

1 Use the Service Browser to determine the new name or location of the service called by the transformer.
2 In the Service Browser, select the flow service containing the transformer you want to rename.
3 In the Flow Editor, select the MAP step containing the transformer. Then, on the Pipeline tab, select the transformer you want to rename.
4 Right-click the transformer and select Properties.
   –OR–
   On the Edit menu, select Properties.
5 In the Transformer Properties dialog box, in the service field, delete the old name, and type in the service’s new fully qualified name in the folderName:serviceName format. Click OK.

Debugging Transformers

When you test and debug a flow service, you can use the following testing and debugging techniques with transformers:

Step into a MAP step and step through the execution of each transformer. For more information about stepping into and out of a MAP step, see “Using the Step Tools with a MAP Step” on page 266.

Set a breakpoint on a transformer so that service execution stops when the transformer is encountered. For more information about setting breakpoints, see “Setting Breakpoints” on page 266.
Disable a transformer so that it does not execute at run time. For more information about disabling transformers, see “Disabling Transformers” on page 271.
Creating SAP BC Schemas, Records, and Specifications

- Creating an SAP BC Schema ......................................................... 190
- Creating a Record ................................................................. 200
- Creating a Specification ........................................................ 206
This chapter provides information about creating, editing, and using SAP BC schemas, records, and specifications.

**Creating an SAP BC Schema**

An SAP BC schema is a “free-standing” element in the Service Browser that acts as the blueprint or model against which you validate an XML document. The SAP BC schema provides a formal description of the structure and content for a valid instance document (the XML document). The formal description is created through the specification of constraints. An SAP BC schema can contain the following types of constraints:

- **Structural constraints** in an SAP BC schema describe the elements, attributes, and types that appear in a valid instance document. For example, an SAP BC schema for a purchase order might specify that a valid `<lineItem>` element must consist of the `<itemNumber>`, `<size>`, `<color>`, `<quantity>`, and `<unitPrice>` elements in that order.

- **Content constraints** in an SAP BC schema describe the type of information that elements and attributes can contain in a valid instance document. For example, the `<quantity>` element might be required to contain a value that is a positive integer.

During data validation, the validation engine in SAP BC Server compares the elements and attributes in the instance document with the structural and content constraints described for those elements and attributes in the SAP BC schema. The validation engine considers the instance document to be valid when it complies with the structural and content constraints described in the SAP BC schema. For more information about data validation, see Chapter 9, “Performing Data Validation” on page 209.

You can create SAP BC schemas from an XML Schema, a DTD (Document Type Definition), or an XML document that references an existing DTD. For information about creating SAP BC schemas, see “Creating an SAP BC Schema” on page 195.

**What Does an SAP BC Schema Look Like?**

The appearance and content of an SAP BC schema depends on whether you generate an SAP BC schema from an XML Schema or a DTD. For example, if you create an SAP BC schema from an XML Schema, the resulting SAP BC schema displays type definitions, element declarations, and attribute declarations. If you create an SAP BC schema from a DTD, the resulting SAP BC schema displays element type declarations.

When you select an SAP BC schema in the Service Browser, Developer displays the contents of the SAP BC schema on the **Schema** tab. The **Schema** tab is divided into two areas: the Schema Browser on the left and the Schema Details area on the right. The **Schema** tab also identifies the target namespace for the SAP BC schema at the top of the tab.
Schema Browser

The Schema Browser displays the components of an SAP BC schema in a format that mirrors the structure and content of the source file. The Schema Browser groups the global element declarations, attribute declarations, simple type definitions and complex type definitions from the source file under the top-level headings ELEMENTS, ATTRIBUTES, SIMPLE TYPES, and COMPLEX TYPES. For example, the ELEMENTS heading contains all of the global element declarations from the XML Schema or the DTD.

If the source file does not contain one of these global components, the corresponding heading is absent. For example, if you create an SAP BC schema from an XML Schema that does contain any global attribute declarations, the Schema Browser does not display the ATTRIBUTES heading. An SAP BC schema created from a DTD never displays the SIMPLE TYPES or COMPLEX TYPES headings because DTDs do not contain type definitions.

Note: A DTD does contain attribute declarations. However, the Schema Browser does not display the ATTRIBUTES heading for SAP BC schemas generated from DTDs. This is because an attribute declaration in a DTD associates the attribute with an element type. Accordingly, the Schema Browser displays attributes as children of the element type declaration to which they are assigned. For more information, see “Element Type Declarations”.

Specifies the target namespace to which the schema belongs.

Select a component in the Schema Browser...

...to view and/or edit the component in the Schema Details area.
The Schema Browser uses unique symbols to represent the components of the SAP BC schema. Each of these symbols relates to a component of an XML Schema or a DTD. The following table identifies the symbol for each component that can appear in an SAP BC schema.

Note: In the following table, global refers to elements, attributes, and types declared or defined as immediate children of the <schema> element in an XML Schema. All element type declarations in a DTD are considered global declarations.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="symbol" alt="Element declaration" /></td>
<td><strong>Element declaration.</strong> An element declaration associates an element name with a type definition. This symbol corresponds to the &lt;element&gt; declaration in an XML Schema and the ELEMENT declaration in a DTD.</td>
</tr>
<tr>
<td><img src="symbol" alt="Element reference" /></td>
<td><strong>Element reference.</strong> An element reference is a reference from an element declaration in a content specification to a globally declared element.</td>
</tr>
<tr>
<td><img src="symbol" alt="Any element declaration" /></td>
<td><strong>Any element declaration.</strong> In XML Schema, an &lt;any&gt; element declaration is a wildcard declaration used as a placeholder for one or more undeclared elements in an instance document.</td>
</tr>
<tr>
<td><img src="symbol" alt="Attribute declaration" /></td>
<td><strong>Attribute declaration.</strong> An attribute declaration associates an attribute name with a simple type definition. This symbol corresponds to the XML Schema &lt;attribute&gt; declaration or the attribute in a DTD ATTLSAP BCT declaration.</td>
</tr>
<tr>
<td><img src="symbol" alt="Attribute reference" /></td>
<td><strong>Attribute reference.</strong> An attribute reference is a reference from a complex type definition to a globally declared attribute. This symbol corresponds to the ref=&quot;globalAttributeName&quot; attribute in an attribute declaration.</td>
</tr>
</tbody>
</table>

DTDs do not have attribute references. Consequently, attribute references do not appear in SAP BC schemas generated from DTDs.
Any attribute declaration. An any attribute declaration is a wildcard declaration used as a placeholder for undeclared attributes in an instance document. This symbol corresponds to the <anyAttribute> declaration in an XML Schema.

Because an <anyAttribute> declaration does not specify an attribute name, the Schema Browser uses 'Any' as the name of the attribute.

Simple type definition. A simple type definition specifies the data type for a text-only element or an attribute. Unlike complex type definitions, simple type definitions cannot carry attributes. This symbol corresponds to the <simpleType> element in an XML Schema.

If the simple type definition is unnamed (an anonymous type), the Schema Browser displays 'Anonymous' as the name of the complex type definition.

Complex type definition. A complex type definition defines the structure and content for elements of complex type. (Elements of complex type can contain child elements and carry attributes.) This symbol corresponds to the <complexType> element in an XML Schema.

If the complex type definition is unnamed (an anonymous type), the Schema Browser displays 'Anonymous' as the name of the complex type definition.

Sequence content model. A sequence content model specifies that the child elements in the instance document must appear in the same order in which they are declared in the content model. This symbol corresponds to the <sequence> compositor in an XML Schema or a sequence list in an element type declaration in a DTD.

Choice content model. A choice content model specifies that only one of the child elements in the content model can appear in the instance document. This symbol corresponds to the <choice> compositor in an XML Schema or a choice list in a DTD element type declaration.

All content model. An all content model specifies that child elements can appear once, or not at all, and in any order in the instance document. This symbol corresponds to the <all> compositor in an XML Schema.
**Schema Details Area**

The Schema Details area displays information that you use to examine and edit the selected component in the Schema Browser. The contents of the Schema Details area vary depending on what component you select. For example, when you select a globally declared element of complex type, the Schema Details area looks like the following.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗</td>
<td><strong>Mixed content.</strong> Elements that contain mixed content allow character data to be interspersed with child elements. This symbol corresponds to the <code>mixed=&quot;true&quot;</code> attribute in an XML Schema complex type definition or a DTD element list in which the first item is <code>#PCDATA</code>.</td>
</tr>
<tr>
<td>🟠</td>
<td><strong>Empty content.</strong> In an XML Schema, an element has empty content when its associated complex type definition does not contain any element declarations. An element with empty content may still carry attributes. In a DTD, an element has empty content when it is declared to be of type <code>EMPTY</code>.</td>
</tr>
</tbody>
</table>

If you select an element declaration...

...the Schema Details area displays information about that element.
When you select a simple type definition, the Schema Details area looks like the following:

**Schema Details area for a Simple Type Definition**

If you select a simple type definition... the Schema Details area displays fields for viewing/editing the simple type.

**Creating an SAP BC Schema**

In Developer, you can create SAP BC schemas from XML Schema definitions, DTDs, and XML documents that reference an existing DTD. The resulting SAP BC schema contains all of the defined types, declared elements, and declared attributes from the source file.

**Note:** The actual work of creating an SAP BC schema is performed by the schema processor. The schema processor is the subsystem of the SAP BC Server that compiles an SAP BC schema from a source file.

**Note:** You can find sample XML Schema definitions in the following directory: `<sapbc>\developer\samples\xml\xsd`. You can also find XML Schema definitions and DTDs on the Web sites for these groups: [www.w3c.org](http://www.w3c.org) and [www.openapplications.org](http://www.openapplications.org).
To create an SAP BC schema

1. Click on the Service Browser toolbar.
2. In the New dialog box, select Schema, and click Next.
3. In the New Schema dialog box, next to Folder, select the folder where you want to save the SAP BC schema.
4. In the Name field, type a name for the SAP BC schema using any combination of letters, numbers, and the underscore character. Click Next.

Important! Developer does not permit the use of certain reserved words and characters that are used in Java or C/C++ (such as for, while, and if) as names. Developer also prohibits the use of a digit as the first character in a name. If you specify a name that uses a reserved word or character, SAP BC Server displays an error message. When this happens, use a different name or try adding a letter or number to the name to make it valid.

5. In the New Schema dialog box, select one of the following to specify the source for the SAP BC schema.

<table>
<thead>
<tr>
<th>Specify...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>Create an SAP BC schema based on an existing DTD referenced by an XML document.</td>
</tr>
<tr>
<td>DTD</td>
<td>Create an SAP BC schema based on a DTD.</td>
</tr>
<tr>
<td>XML Schema</td>
<td>Create an SAP BC schema based on an XML Schema definition.</td>
</tr>
</tbody>
</table>

Important! You can create an SAP BC schema from an XML document only if the XML document references an existing DTD.

6. Click Next.

7. In the New Schema dialog box, under Enter a URL or select a local file, do one of the following:
   - If you want to base the SAP BC schema on an XML document, DTD, or XML Schema definition that resides on the Internet, type the URL of the resource. (The URL you specify must begin with http: or https:.)
   - If you want to base the SAP BC schema on an XML document, DTD, or XML Schema definition that resides on your local file system, type the path and file name, or click to navigate to and select the file.
8 Click Finish. Developer generates the SAP BC schema using the document you specified and displays it in the Editor in the Developer window.

**Note:** You might receive errors or warnings when creating an SAP BC schema from an XML Schema definition or DTD. For more information about these errors and warnings, see “Validation Errors and Exceptions” on page 627.

**Note:** When creating an SAP BC schema from an XML Schema definition, Developer validates the schema and does not create the SAP BC schema if the XML Schema definition is not valid. For more information, see “SAP BC Schema Generation Errors and Warnings” on page 643.

**Creating SAP BC Schemas from XML Schemas that Reference Other Schemas**

A schema author can insert the elements, attributes, and type definitions from another schema into the schema they are creating. A schema author might do this to break up a large XML Schema into several small, more reusable XML Schemas. When you generate an SAP BC schema from an XML Schema that references another schema, the schema processor either includes all of the schema components in a single SAP BC schema, creates multiple SAP BC schemas, or creates no SAP BC schema at all. The behavior of the schema processor depends on the mechanism the source XML Schema uses to reference the other schema. The following mechanisms can be used to reference an external schema:

- **Include.** When you generate an SAP BC schema from an XML Schema that uses `<include>` to include the contents of an external schema in the same namespace, the resulting SAP BC schema contains all of the defined types, declared elements, and declared attributes from the source schema and the external schema.

- **Import.** When you generate an SAP BC schema from an XML Schema that contains an `<import>` element to import the contents of an external schema in a different namespace, the schema processor creates one SAP BC schema per namespace. For example, if the source XML Schema imports two XML Schemas from the same namespace, the schema processor creates an SAP BC schema for the source XML Schema and then a second SAP BC schema that includes the components from the two imported XML schemas. The schema processor assigns each imported schema the name that you specify and appends an underscore and a number to each name. For example, if you create an SAP BC schema named “mySchema” from mySchema.xsd, the schema processor generates an SAP BC schema named “mySchema_2” for the imported XML Schema.

- **Redefine.** Schema authors can also use `<redefine>` to include and then redefine type definitions, model groups, and attribute groups from an external XML Schema in the same namespace. The schema processor in SAP BC Server does not support `<redefine>` at this time. The schema processor will not generate an SAP BC schema for an XML Schema that contains a `<redefine>` mechanism. When the schema
processor encounters a <redefine> mechanism in an XML Schema, Developer displays the following message:

Detected use of "redefine" in {XML Schema Name}. SAP BC Server does not support "redefine".

### Editing a Simple Type in an SAP BC Schema

You can modify a simple type definition in an SAP BC schema without editing the source XML Schema definition. You edit a simple type by adding or changing the value of one or more constraining facets applied to the simple type. For example, you can modify a simple type by adding an enumerated value, a pattern constraint, or changing the length constraint. Editing the simple type through Developer is an alternative to editing the source XML Schema definition and regenerating the SAP BC schema.

You can edit any of the globally defined simple types (those that appear under the SIMPLE TYPES heading) or anonymous simple types (those defined as part of an element or attribute declaration.) A named simple type that appears as a child of an element or attribute in the Schema Browser cannot be edited. (These simple types are global simple types used to define the element or attribute they appear under.) The following illustration identifies the simple type definitions that you can and cannot edit.

### Editable simple type definitions in an SAP BC schema

[Diagram showing editable and non-editable simple type definitions]
Creating an SAP BC Schema

When modifying a simple type definition, keep the following points in mind:

- Changes to a simple type definition affect the elements and attributes for which the simple type is the defined type. For example, if the attribute `partNum` is defined to be of type `SKU`, changes to the `SKU` simple type definition affect the `partNum` attribute.

- Changes to a global simple type definition in an SAP BC schema do not affect simple types derived from the global simple type; that is, the changes are not propagated to the derived types in the SAP BC schema.

- Simple types in an SAP BC schema can be used as content constraints for variables in pipeline validation. Consequently, changes to a simple type also affect every variable to which the simple type is applied as a content constraint.

Tip! You can create a custom simple type to apply to a variable as a content type constraint. For more information about creating a custom simple type and applying constraints to variables, see “Setting Constraining facet Values” on page 200.

- Changes to a simple type definition are saved in the SAP BC schema. If you regenerate the SAP BC schema from the XML Schema definition, your changes will be overwritten.

- When you edit the constraining facets applied to a simple type definition, you can only make the constraining facet values more restrictive. The constraining facets cannot become less restrictive. For more information about setting values for constraining facets, see “Setting Constraining facet Values” on page 200.

Tip! If you want to edit complex type definitions, attribute declarations, element declarations, or the structure of the schema, you need to edit the XML Schema and then regenerate the SAP BC schema.

To edit a simple type definition

1. In the Service Browser, select the SAP BC schema that contains the simple type you want to edit.

2. In the Schema Browser, select the simple type that you want to edit. The  symbol appears next to simple types.

3. In the Schema Details are, specify the constraining facets that you want to apply to the simple type. For more information about constraining facets, see “Constraining Facets” on page 623.

4. To save your changes, click  on the Service Browser toolbar.
CHAPTER 8 Creating SAP BC Schemas, Records, and Specifications

Setting Constraining Facet Values

You can edit any of the constraining facet values that appear in the Schema Details area when you select an editable simple type definition in the Schema Browser. The constraining facets displayed in the Schema Details area depend on the primitive type from which the simple type was derived. For example, if the simple type definition is derived from string, the Schema Details area displays the `enumeration`, `length`, `minLength`, `maxLength`, `pattern`, and `whiteSpace` facets. The Schema Details area only displays constraining facet values set in the simple type definition—it does not display constraining facet values the simple type definition inherited from the simple types from which it was derived.

You can view the constraining facet values set in the type definitions from which a simple type was derived by clicking the `Base Constraints` button. Base constraints are the constraining facet values set in all the type definitions from which a simple type is derived—from the primitive type to the immediate parent type. These constraint values represent the cumulative facet values for the simple type.

When you edit the constraining facets for a simple type definition, you can only make the constraining facets more restrictive—the applied constraining facets cannot become less restrictive. For example, if the `length` value is applied to the simple type, the `maxLength` or `minLength` values cannot be set because the `maxLength` and `minLength` facets are less restrictive than `length`.

Creating a Record

A record in the Service Browser can be used to specify input or output parameters for a service or specification. A record can also be used to build a Record or Record List variable and as the blueprint for pipeline validation and record validation.

Records can provide the following benefits:

- Using a record as the input or output signature for a service can reduce the effort required to build a flow.
- Using a record to build Record or Record List variables can reduce the effort needed to declare input or output parameters or the effort/time needed to build other records.
- Records improve accuracy, because there is less opportunity to introduce a typing error typing variable names.
- Records make future changes easier to implement, because you can make a change in one place—the record—rather than everywhere the record is used.

If you make a change to a record, keep in mind that any change is automatically propagated to all services, specifications, Record variables, and Record List variables that use or reference the record. (This happens when you save the updated record to the server.)
Creating a Record

**Important!** If you use a record as the blueprint for XML, pipeline, or record validation, any changes you make to the record can affect whether the object being validated (XML document, pipeline, or record) is considered valid. For more information about validation, see “Performing Data Validation” on page 209.

You can create an empty record in which you define the structure and the variables, or you can base the record on an XML document, DTD, XML Schema definition, SAP DDIC Structure or SAP IDoc. When you create a record using one of these elements, Developer creates a record with the same structure and same variable constraints as the source document.

### Creating a Record from an XML Schema Definition or a DTD

When you use an XML Schema definition, DTD, or an XML document that references a DTD as the source for a record, Developer creates a record and an SAP BC schema. The SAP BC schema contains the elements, attributes, and data types contained in the XML Schema definition. The record, which displays the variables and structure of the document, uses links to the SAP BC schema to obtain content type information about named simple types.

When you generate a record from an XML Schema definition, Developer also validates the XML Schema definition before creating the SAP BC schema or the record. If the XML Schema definition does not conform syntactically to the schema for XML Schemas defined in XML Schema Part 1: Structures, Developer does not create an SAP BC schema or a record. Instead, Developer displays an error message that lists the validation errors and the location of the errors within the XML Schema definition.

**To create a record**

1. **On the File menu, click New.** The wizard starts.
2. **In the New dialog box, select Record and click Next.**
3. **In the New Record dialog box, do the following:**
   1. **In the list next to Folder, select the folder in which you want to save the record.**
   2. **In the Name field, type a name for the record using any combination of letters, numbers, and/or the underscore character.**

**Important!** Developer does not allow the use of certain reserved words and characters that are used in Java or C/C++ (such as for, while, and if) as names. If you specify a name that is a reserved word, Developer displays an error message. When this happens, use a different name or try adding a letter or number to the name you have specified to make it valid.
In the New Record dialog box, select one of the following to specify whether you want to build a record based on the structure and elements defined in an existing XML document, DTD, XML Schema, SAP DDIC Structure or SAP IDoc:

- **None**: Create an empty record in which you define variables.
- **XML**: Create a record that matches the structure and variables in an XML document.
- **DTD**: Create a record that matches the structure and variables in a DTD.
- **XML Schema**: Create a record that matches the structure and variables in an XML Schema definition.
- **SAP Structure**: Create a record that matches a structure of the SAP DDIC (data dictionary).
- **SAP IDoc**: Create a record that matches the structure and segments of an SAP IDoc.

If you selected **None**, click **Finish**, and skip to step 8.

Otherwise, click **Next**.

In the New Record dialog box, do one of the following:

- If you want to base the record on an XML document, DTD, or XML Schema definition that resides on the Internet, type the URL of the resource. (The URL you specify must begin with `http:` or `https:`.)
- If you want to base the record on an XML document, DTD, or XML Schema definition that resides on your local file system, type in the path and file name, or click to navigate to and select the file.
- If you want to base the record on an SAP Structure or an SAP IDoc, select the SAP system you want to retrieve the information from and enter the name of the structure or IDoc you are looking for.

Do one of the following:

- If you selected **XML** in step 4, click **Finish**. Developer generates the record using the XML document you specified and displays it on the Record tab. If you need to modify the generated record, proceed to the next step. Otherwise, skip to step 9.
- If you selected **SAP Structure** or **SAP IDoc** in step 4, click **Finish**. Developer generates the record using the structure or IDoc you specified and displays it on the Record tab. If you need to modify the generated record, proceed to the next step. Otherwise, skip to step 9.
— If you selected DTD or XML Schema in step 4, click Next. Developer prompts you to select the root element for the document. Select the root element and click OK. Developer generates the record and displays it on the Record tab. Developer also generates an SAP BC schema and displays it in the Service Browser. If you need to modify the generated record, proceed to the next step. Otherwise, skip to step 9.

Note: You might receive errors or warnings when creating a record from a DTD or XML Schema definition. For more information about these errors and warnings, see “Validation Errors” on page 628.

8 If you want to add or edit variables in the record, click the Record tab to make it active. For each variable in the record, do the following:

1. Click on the toolbar and select the type of variable that you want to specify.
2. Type the name of the variable and then press ENTER.
3. Right-click the variable and select Properties to set variable properties and apply validation constraints (optional).
   For more information about setting variable properties, see “Specifying Variable Properties” on page 205. For details on applying constraints, see “Applying Constraints to Variables” on page 211.
4. If the variable is a Record or a Record List, repeat steps 1–3 to define its member variables. Then use to indent each member variable beneath the Record or Record List variable.

9 On the File menu, click Save (or click in the Service Browser). The record is saved to the SAP BC Server.

Printing a Record

You can use the View as HTML command to produce a printable version of a record. This lets you see all the variables in a record in a single document.

To print a record

1. In the Service Browser, select the record you want to print.
2. From the File menu, select View as HTML or click .
   Developer expands any Record and Record List variables in the record, creates an HTML page containing the record, and displays the HTML page in your default browser.
3. If you want to print the record, select your browser’s print command.
CHAPTER 8 Creating SAP BC Schemas, Records, and Specifications

Using a Record to Specify Service Input or Output Parameters

You can use a record as the set of input or output parameters for a service or specification. If you have multiple services with identical input parameters but different output parameters, you can use a record to define the input parameters rather than manually specifying individual input variables for each service.

To use a record as service input or output parameters

1. In the Service Browser, select the service to which you want to assign the record.
2. Click the Input/Output tab.
3. Under Input or Output (depending on which half of the Input/Output tab you want to apply the record to), type the fully qualified name of the record or click to select it from a list.
4. On the File menu, click Save (or click in the Service Browser).

Important! When a record is assigned to the input or output of a service, you cannot add, delete, or modify the variables on that half of the Input/Output tab.

Using a Record to Build a Record Reference or Record Reference List Variable

You can use a record to build a “Record Reference” or “Record Reference List” variable. By referencing an existing record instead of creating a new one, you can reduce the time required to create variables and maintain better consistency for variable names. You might find referencing variables especially useful for information that is repeated over and over again, such as address information.

To use a record to build a Record or Record List variable

1. On the Input/Output tab, click on the toolbar and select one of the following:

<table>
<thead>
<tr>
<th>Click...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record Reference</td>
<td>Create a Record variable based on a record in the Service Browser.</td>
</tr>
<tr>
<td>Record Reference List</td>
<td>Create a Record List variable based on a record in the Service Browser.</td>
</tr>
</tbody>
</table>

2. In the Name field, type the fully qualified name of the record or select it from the list next to Folder.
3 Click OK.
4 Type the name of the variable.
5 Press ENTER.

**Important!** If you use a record to build a Record or Record List variable, you cannot directly add, delete, or modify members of that variable on the Input/Output tab. To edit the referenced Record or Record List, select it in the Service Browser, and then edit the variables on the Record tab.

**Specifying Variable Properties**

For certain variable types, such as String, you can specify additional properties of the variable. These properties are located in the Variable Properties dialog box, which is accessed via Edit ➤ Properties or the right-click menu.

### Variable Properties Dialog Box

On the Variable Properties dialog box, the General tab contains the properties of the variable; the Constraints tab contains validation constraints for the variable. (For more information about specifying validation constraints, see “Applying Constraints to Variables” on page 211.)

The Text Field, Password, Large Editor, and Pick List options affect how you input data for the variable as follows.
Creating a Specification

A specification is a “free-standing” SAP BC element that defines a set of service inputs and outputs. If you have multiple services with the same input and output requirements, you can point each service to a single specification rather than manually specifying individual input and output variables in each service.

Using specifications provides the following benefits:

- It reduces the effort required to build each flow.
- It improves accuracy, because there is less opportunity to introduce a typing error when defining a variable name.
- It makes future specification changes easier to implement, because you can make the change in one place—the specification—rather than in each individual service.

If you use a specification, keep in mind that:

- Any change that you make to the specification is automatically propagated to all services that reference that specification. (This happens the moment you save the updated specification to the server.)
- A specification wholly defines the input and output parameters for a service that references it. This means that you cannot directly alter the service’s input and output parameters through its Input/Output tab. (Developer displays the parameters, but does not allow you to change them.) To make changes to the input and output parameters of the service, you must modify the specification (which affects all services that reference it) or detach the specification so you can manually define the parameters on the service’s Input/Output tab.

You create a specification with Developer. You assign a specification to a service using the Input/Output tab for the service.

2. In the New dialog box, select Specification, and click Next.

3. In the New Specification dialog box, do the following:
   1. In the list next to Folder, select the folder to which you want to save the specification.
   2. In the Name field, type a name for the specification using any combination of letters, numbers, and the underscore character.

   **Important!** Developer does not allow the use of certain reserved words (such as for, while, and if) as names. If you specify a name that is a reserved word, you will receive an error message. When this happens, use a different name or try adding a letter or number to the name you specified to make it valid.

3. Click Finish.

4. If you want this specification to reference another specification, do the following:
   1. In Specification Reference field, type the specification's name or click to select it from a list.
   2. Skip the rest of this procedure.

   **Important!** Typically, you do not build a specification by referencing another specification. However, it is useful to do this in the situation where you will use the specification with a group of services whose requirements are expected to change (i.e., they match an existing specification now, but are expected to change at some point in the future). Referencing a specification gives you the convenience of using an existing specification and the flexibility to change the specification for only that single group of services in the future.

5. If you want to reference a record for the Input or Output half of the specification. Do the following:
   1. In the Input or Output field (depending on which half of the specification you want to assign the record to) type the record name or click to select it from a list. For information about creating records, see “Creating a Record” on page 200.
   2. Proceed to the next step to specify the variables for the other half of the specification. (If you assigned records to both the Input and Output sides of this specification, skip the rest of this procedure.)

   **Important!** Once you assign a record to the Input or Output side of a specification, you cannot add, delete, or modify the variables on that half of the Input/Output tab.
For each input or output variable that you want to define, do the following:

1. Select the half of the specification (Input or Output) where you want to define the variable by clicking anywhere in that half’s large white text box.
2. Click on the toolbar and select the type of variable that you want to specify.
3. Type the name of the variable and then press ENTER.
4. Right-click the variable and select Properties to set variable properties and apply validation constraints (optional).
   For details on setting variable properties, see “Specifying Variable Properties” on page 205. For details on applying constraints, see “Applying Constraints to Variables” on page 211.
5. If the variable is a Record or a Record List, repeat steps 2–4 to define each of its members. Then use to indent each member beneath the Record or Record List variable.

On the File menu, click Save (or click in the Service Browser). The specification is saved to the SAP BC Server.

### To assign a specification to a service

1. In the Service Browser, select the service to which you want to assign a specification.
2. Click the Input/Output tab.
3. In Specification Reference, type the fully qualified name of the specification, or click to select it from a list.

**Important!** When a specification is assigned to a service, you cannot add, delete, or modify the declared variables using that service’s Input/Output tab.
Performing Data Validation

- What Is Data Validation? ................................................. 210
- Applying Constraints to Variables ........................................ 211
- Performing XML Validation in SAP BC Server ......................... 218
- Performing Pipeline Validation ........................................ 222
- Performing Input/Output Validation ..................................... 226
- Performing Record and Node Validation ............................... 229
- Validation Errors and Exceptions ..................................... 233
What Is Data Validation?

Data validation is the process of verifying that run-time data conforms to a pre-defined structure and format. Data validation also verifies that run-time data is a specific data type and falls within a defined range of values.

By performing data validation, you can make sure that:

- The pipeline, a record, or an XML document contains the data needed to execute subsequent services. For example, if a service processes a purchase order, you might want to verify that the purchase order contains a customer name and address.

- The data is in the structure expected by subsequent services. For example, a service that processes a purchase order might expect the customer address to be a record variable with the following fields: name, address, city, state, and zip.

- Data is of the type and within a value range expected by a service. For example, if a service processes a purchase order, you might want to make sure that the purchase order does not contain a negative quantity of an item (such as -5 shirts).

By using the data validation capabilities built into SAP BC Server, you can decide whether or not to execute a service based on the validity of data. The validation capabilities can also eliminate extra validation code from your services.

What Is Data Validated Against?

During validation, run-time data is compared to a blueprint or model. The blueprint or model is a formal description of the structure and the allowable content for the data. The blueprint identifies structural and content constraints for the data being validated. The validation engine in SAP BC Server considers the data to be valid when it conforms to the constraints specified in the blueprint. A blueprint can be an SAP BC schema, record, or a set of input and output parameters.

The blueprint used to validate data depends on the type of validation you are performing. In SAP BC Server, you can perform the following types of validation:

- **XML validation.** The validation engine in SAP BC Server validates the structure and content of an XML document against an SAP BC schema.

- **Pipeline validation.** The validation engine in SAP BC Server validates the structure and content of the pipeline against a record.

- **Input/Output validation.** The validation engine in SAP BC Server validates the input and/or output of a service against the declared input and/or output parameters declared for the service.

- **Record and node validation.** The validation engine in SAP BC Server validates the structure and content of an individual record or Document in the pipeline against a record or an SAP BC schema.
The following sections provide information about performing each type of validation and information about applying constraints to variables and records.

Applying Constraints to Variables

In pipeline, record, and input/output validation, the blueprint used for data validation needs to have constraints applied to its variables. Constraints are the restrictions on the structure or content of variables in the record used to validate the pipeline. You can apply the following types of constraints to variables:

- **Structural** constraints specify the existence and structure of variables at run time. For example, if the flow service in which you are validating the pipeline processes a purchase order, you might want to check that values for the `purchaseOrderNumber`, `accountNumber`, and `customerName` variables exist at run time.

  For Record and Record List variables, you can also specify the structure of the variable—that is, you can specify what variables can be contained in the Record or Record List at run time. For example, you could specify that the `lineItem` Record variable must contain the child variables `itemNumber`, `quantity`, `size`, `color`, and `unitPrice`. You could also specify that the `lineItem` Record can optionally contain the child variable `specialInstructions`.

- **Content** constraints describe the data type for a variable and the possible values for the variable at run time. When you specify a content constraint for a variable, you can also apply a constraining facet to the content. A constraining facet places limitations on the content (data type). For example, for a String variable named `itemQuantity`, you might apply a content constraint that requires the value to be an integer. You could then apply constraining facets that limit the content of `itemQuantity` to a value between 1 and 100.

  You can use simple types from an SAP BC schema as content constraints for variables. You can only apply content constraints to String, String List, or String Table variables.

For pipeline and record validation, the record used as the blueprint needs to have constraints applied to its variables. For input/output validation, constraints need to be applied to the declared input and output parameters of the service being validated.

**Note:** When you create a record from an XML Schema definition or a DTD, the constraints applied to the elements and attributes in the original document are preserved in the new record. For more information about creating records, see “Creating a Record” on page 200.
To apply constraints to a variable

1. Select the variable to which you want to apply constraints.
   You can apply constraints to variables in records, variables in a specification, and variables declared on the Input/Output tab.

2. Right-click the variable and select Properties. Developer opens the Variable Properties dialog box.

3. In the Variable Properties dialog box, click the Constraints tab.

Constraints tab

4. If you want to require the selected variable to exist at run time, select the Field must exist at run-time check box. If the existence of the variable is optional, leave the check box cleared.
Applying Constraints to Variables

5 If the selected variable is a Record or Record List, and you want to allow it to contain child variables that are not specified in the record, select the **Allow unspecified fields** check box.

If the **Allow Unspecified Fields** check box is cleared, at run time, any child variables that exist in the pipeline but do not appear in the record will be treated as errors.

6 If the selected variable is a String, String List, or String Table, and you want to specify content constraints for the variable, do one of the following:

   — If you want to use a content type that corresponds to a simple type built-in to XML Schema, in the **Content type** list, select the type for the variable contents. For a description of these content constraints, see Appendix G, “Validation Content Constraints” on page 611.

   — If you want to use a simple type from an SAP BC schema as the content constraint, click the **Browse** button. In the **Browse** dialog box, select the SAP BC schema containing the simple type you want to apply. Then, select the simple type you want to apply to the variable.

   **Note:** A content type corresponds to a simple type from an XML Schema definition. All of the choices in the **Content type** list correspond to simple types defined in XML Schema Part 2: Datatypes.

7 Do one of the following:

   — If you want to customize the content type by changing the constraining facets applied to the type, see “Customizing a Content Type” on page 213.

   — If you want to apply the selected content type to the variable, click **OK**.

8 Repeat steps 2–7 for each variable to which you want to apply constraints in the record, specification, service input, or service output.

9 Click to save your changes.

**Customizing a Content Type**

Instead of applying an existing content type or simple type to a variable, you can customize an existing type and apply the new, customized type to a variable. You customize a content type or simple type by changing the constraining facets applied to the type.

When you customize a type, you actually create a new content type. Developer saves the changes as a new content type named `contentType_customized`. For example, if you customize the string content type, Developer saves the new content type as `string_customized`. 
When customizing a content type, keep the following points in mind:

- When you edit the constraining facets applied to a content type, you can only make the constraining facet values more restrictive. The constraining facets cannot become less restrictive. For more information about setting values for constraining facets, see “Constraining Facets” on page 623.

- The constraining facets you can specify depend on the content type. For more information about the constraining facets for each content type, see Appendix C, “Supported Data Types” on page 577.

- The customized content type applies only to the selected variable. To make changes that affect all variables to which the content type is applied, edit the content type in the SAP BC schema. (All content types are simple types from SAP BC schemas.) For more information about editing simple types, see “Editing a Simple Type in an SAP BC Schema” on page 198.

---

**Note:** When you edit a simple type definition in an SAP BC schema, the changes are saved to the selected type definition in the SAP BC schema. The modifications affect any variable to which the simple type is applied as a content type constraint. For more information about editing a simple type definition, see “Editing a Simple Type in an SAP BC Schema” on page 198.

To customize a content type

1. Select the variable to which you want to apply a customized content type.
2. Right-click the variable, and select Properties.
3. In the Variable Properties dialog box, click the Constraints tab.
4. To select the content type you want to customize, do one of the following:
   - In the Content type list, select the content type you want to customize.
   - If you want to customize a simple type from an SAP BC schema, click the Browse button. In the Browse dialog box, select the SAP BC schema containing the simple type. Then, select the simple type you want to customize and apply to the variable.
5. Click the Customize button. Developer makes the constraining facet fields below the Content type list available for data entry. (Developer changes the background of the constraining facet fields from grey to white.) Developer changes the name of the content type to contentType_customized.
In the fields below the **Content type** list, specify the constraining facet values you want to apply to the content type.

<table>
<thead>
<tr>
<th>In this field...</th>
<th>Specify...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>enumeration</strong></td>
<td>The possible values for the variable at run time. Click <img src="image" alt="Icon" /> to insert the possible values.</td>
</tr>
</tbody>
</table>

**Note:** If you also entered values using the Pick List feature on the **General** tab, those values will be displayed at run time. However, the **enumeration** constraints will be used for validation.

- **fractionDigits**: The maximum number of digits to the right of the decimal point. For example, 99.999 has a **fractionDigits** value of 2.
- **length**: The precise units of length required for the variable value.
- **minLength**: The minimum units of length permitted for the variable value.
- **maxLength**: The maximum units of length permitted for the variable value.
- **minInclusive**: The lower bound of a range of possible values, including the value you specify. The variable can have a value greater than or equal to **minInclusive**.
- **minExclusive**: The lower bound of a range of possible values, not including the value you specify. The variable can have a value greater than but not equal to **minExclusive**.
- **maxInclusive**: The upper bound of a range of possible values, including the value you specify. The variable can have a value less than or equal to **maxInclusive**.
- **maxExclusive**: The upper bound of a range of possible values, not including the value you specify. The variable can have a value less than but not equal to **maxExclusive**.
- **pattern**: A pattern (regular expression) that the value of the variable must match. For example, you can use a regular expression to specify that a variable that is a string content constraint must match a Social Security number format.
- **totalDigits**: The maximum number of decimal digits allowed in a value. For example, the **totalDigits** of the value 999.99 is 5.
Click OK. Developer saves the changes as a new content type named contentType_customized.

**Important!** Developer throws exceptions at design-time if the constraining facet values for a content type are not valid. For more information about these exceptions, see Appendix H, “Validation Errors and Exceptions” on page 627.

### Viewing the Constraints Applied to Variables

The appearance of a variable name in a record or on the Input/Output tab depends on the types of constraints that are applied to the variable. Optional variables appear in italic. Variables to which content constraints have been applied appear in bold. The following illustration shows the constraints applied to variables in the record addressBlock.
Applying Constraints to Variables

Constraints applied to variables in a record

Referencing Other Records

If you find that certain pieces of information (for example, an address) are repeated in the records and input and output parameters, you can create a record that contains the repeated information. You can then reference that record instead of creating the record, inserting the variables, and applying constraints each time you need to use an address for validation.

For example, the following illustration shows a record that includes references to the addressBlock record in the Service Browser.

A Record that contains Record References
Performing XML Validation in SAP BC Server

In SAP BC Server, XML validation is the process of verifying the structure and content of an XML document against an SAP BC schema. By validating an XML document, you can ensure that the document contains the elements and attributes organized in the structure or format expected by subsequent services. You can also ensure that the elements and attributes contain values that are of the data type expected by subsequent services. In SAP BC Server, an XML document is valid when it complies with the structural and content constraints described in the SAP BC schema it is validated against.

For example, if you receive an XML document that contains new employee information, you might want to validate the employee information before executing subsequent services. You might want to make certain the XML document contains an employee name, an address, telephone number, and date of birth information. Additionally, you might want to validate the date of birth information to make sure that it conforms to a specific date format.

What Is an XML Document Validated Against?

When you validate an XML document, an SAP BC schema acts as the blueprint against which the XML document is validated. The SAP BC schema that acts as the blueprint specifies:

- Which elements are required to appear in the XML document and which elements are optional.
- The structure and order of elements in the XML document.
- Which elements contain child elements and the order of the child elements.
- The type of data that elements and attributes can contain.
- The number of times an element can appear in an XML document.
- Which attributes correspond to which elements.

For information about creating SAP BC schemas, see “Creating an SAP BC Schema” on page 190.

Note: If the XML document has been converted to a record, validate the XML document against a record. For more information about validating a record, see “Creating a Record” on page 200?
Validating an XML Document

To validate an XML document, invoke the pub.schema:validate service. This service instructs the validation engine to validate an XML document by comparing it to a specified SAP BC schema or a record in the Service Browser. The XML document is valid if it complies with the structural and content constraints described in the SAP BC schema or record. The pub.schema:validate service returns a string that indicates whether validation was successful and an errors variable (an IData array) that contains any validation errors.

When you use the pub.schema:validate service to validate an XML document, keep the following points in mind:

- You can specify the maximum number of errors to be collected by the service.
- You can specify whether the pub.schema:validate service should fail if the XML document is invalid.
- If you want to validate an XML document against an SAP BC schema, the XML document needs to be a parsed document (a Document node). You can use the pub.web:loadDocument service or the pub.web:stringToDocument to parse an XML document. You can also submit an XML document directly to the server via HTTP or FTP. When the server receives the document, it will automatically parse it. For more information about submitting XML documents to services, see “Passing XML Data to a Service” on page 371.
- If you want to validate an XML document against a record, the XML document needs to be a record. You can use the pub.web:documentToRecord service to create a record from a Document node.

Note: For more information about the pub.web services, see the SAP BC Built-In Services Guide.

To validate an XML document

1. In the Service Browser, select the flow service in which you want to validate an XML document.

2. In the Flow Editor, click and select Browse.

3. Use the list next to Folder to navigate to and select the pub.schema:validate service and click OK.

4. In the Flow Editor, move the validate service to the point in the flow where you want to validate the XML document.

5. Click the Pipeline tab. The Pipeline Editor for the pub.schema:validate service appears.
6 Map the variable for the XML document from **Pipeline In** to **object** in **Service In**. The XML document needs to be of type Document node or record. The Pipeline Editor displays Document nodes as object variables.

Map the XML document variable (node) to the object variable

7 Under **Service In**, select the **conformsTo** variable and click . Developer opens the **Input for** dialog box.

8 In the **Input for** dialog box, do one of the following:
   - If the XML document is a node (Document), type the fully qualified name of the SAP BC schema that you want to validate the XML document against. For example: `folderName:schemaName`. Click **OK**.
   - If the XML document is a record, type the fully qualified name of the record that you want to validate the XML document against. For example: `folderName:recordName`. Click **OK**.

**Note:** The nominated SAP BC schema is only used to validate unqualified nodes, that is, nodes with XML Namespaces URI = null.

9 Use the following table to assign values to the remaining **Service In** variables for the **pub.schema:validate** service. All of these variables are optional.
For this property... Specify...

maxErrors  Number of errors to be collected. When the number of errors found is equal to `maxErrors`, the validation engine stops validating the XML document and returns the result. If `maxErrors` is set to -1, the server returns all of the errors. Default is 1.

generateContent  Whether the validation engine should ignore content (values of variables).

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Specifies that the validation engine ignores the contents of variables. The validation engine validates the XML document for structural constraints only.</td>
</tr>
<tr>
<td>False</td>
<td>Specifies that the validation engine does not ignore the contents of variables. The validation engine validates the XML document for content and structural constraints.</td>
</tr>
</tbody>
</table>

Default is false.

failIfInvalid  Whether the pub.schema:validate service fails if the XML document is invalid:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Fails the service if the XML document is invalid. When the service fails, an exception is thrown and the errors are embedded in the exception message.</td>
</tr>
<tr>
<td>False</td>
<td>Indicates service success even if the XML document is invalid and returns errors via the pipeline.</td>
</tr>
</tbody>
</table>

Default is false.

10  Click ![ ] to save your changes.

For information about errors that can occur during validation, see “Validation Errors and Exceptions” on page 233.
Performing Pipeline Validation

Pipeline validation is the process of verifying the contents of the pipeline against a record. By validating pipeline data, you can:

- Ensure a higher degree of accuracy for pipeline content.
- Execute or not execute a service based on the validity of the pipeline data.
- Eliminate extra validation code in your service.

In pipeline validation, a record in the Service Browser is the blueprint or model that you validate the pipeline against. The constraints applied to the variables in the record determine what can and cannot be included in the pipeline. For more information about applying constraints to variables, see “Applying Constraints to Variables” on page 211.

After you define and apply constraints to the record that you want to validate the pipeline against, you can validate the pipeline using the built-in service `pub.schema:validatePipeline`. This service instructs the validation engine to validate the pipeline by comparing the pipeline contents against a specified record in the Service Browser. The pipeline is valid when it conforms to the structural and content constraints applied to the record. The `pub.schema:validatePipeline` service returns a string that indicates whether validation was successful and an IData array (errors variable) that contains any validation errors.

When you insert the `pub.schema:validatePipeline` service, you can determine the maximum number of errors to be collected by the server. You can also specify whether the `pub.schema:validatePipeline` service should fail if the pipeline is invalid.

---

To validate the pipeline using the `pub.schema:validatePipeline` service

1. In the Service Browser, select the flow service in which you want to validate the pipeline.
2. Click and select *Browse*.
3. Use the list next to *Folder* to navigate to and select the `pub.schema:validatePipeline` service.
4. In the Flow Editor, move the `validatePipeline` service to the point in the flow where you want to validate the pipeline.
5. Click the *Pipeline* tab.
6. Under *Service In*, select *conformsTo* and click .
7. In the *Input for* dialog box, type the name of the record that you want to validate the pipeline against and click *OK*. You need to type the fully qualified name of the record. For example: `folderName:recordName`.
8. Use the following table to assign values to the remaining *Service In* variables for the `validatePipeline` service.
Performing Pipeline Validation

For this property... Specify...

maxErrors
Number of errors to be collected. When the number of errors found is equal to `maxErrors`, the validation engine stops validating the pipeline and returns the result. If `maxErrors` is set to -1, the server returns all of the errors. Default is 1.

ignoreContent
Whether the validation engine should ignore content (values of variables).

<table>
<thead>
<tr>
<th>True</th>
<th>Instructs the validation engine to ignore the contents of variables in the pipeline. The validation engine validates the pipeline for structural constraints only.</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>Specifies that the validation engine does not ignore the contents of variables. The validation engine validates the pipeline for content and structural constraints.</td>
</tr>
</tbody>
</table>

Default is false.

failIfInvalid
Whether the `pub.schema:validatePipeline` service fails if the pipeline is invalid:

<table>
<thead>
<tr>
<th>True</th>
<th>Fails the service if the pipeline is invalid. When the service fails, an exception is thrown and the errors are embedded in the exception message.</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>Indicates service success (even if the pipeline is invalid) and returns errors via the pipeline.</td>
</tr>
</tbody>
</table>

Default is false.

9 Click to save your changes to the flow service.
Validating the Pipeline via the pub.schema:validatePipeline Service

You can use built-in services pub.schema:validate and pub.schema:validatePipeline to perform pipeline validation from within a Java service. In the following example, the pub.schema:validate service is used to validate the results of the pub.web:documentToRecord service against a specification for an OAG purchase order.
Performing Pipeline Validation

```java
IData validInput;
IData dtrResult;

// initialize the folder and record name to point to a record
// that exists on SAP BC Server
String ifc = "OAG.PO"
String rec = "purchaseOrder"

// put the result from the documentToRecord service (i.e, the object to be
// validated) into the key named <object>
IDataCursor validCursor = validInput.getCursor();
IDataCursor dtrCursor = dtrResult.getCursor();
dtrCursor.first("boundNode");
validCursor.insertAfter("object", dtrCursor.getValue());
dtrCursor.destroy();

// set the <conformsTo> parameter to point to the record to validate against
// This record must exist on SAP BC Server
validCursor.insertAfter("conformsTo", ifc+":"+rec);
validCursor.destroy();

// set the <maxErrors> parameter to the number of allowed validation errors
validCursor.insertAfter("maxErrors", "1000");
validCursor.destroy();

// invoke pub.schema.validate to validate contents of <object>
IData validResult = context.invoke("pub.schema", "validate", validInput);

// check <isValid> to see whether <object> is valid and process accordingly
IDataCursor validCursor = validResult.getCursor();
if(validCursor.first("isValid"))
{
    if (IDataUtil.getString(validCursor).equals("false"))
    {
        IData [] vr = validResult.getValuesArray("errors");
        System.out.println("ERROR(s) found with example");
        for (int j=0; j < vr.length; j++ )
        {
            System.out.println("vr[j].toString()");
        }
    }
    validCursor.destroy();
}
```

SAP BC Developer Guide 4.8
Performing Input/Output Validation

In input/output validation, the validation engine in SAP BC Server validates the inputs and/or outputs of a service against the declared input and output parameters of the service. If you specify that you want to validate the inputs to the service, the validation engine validates the service input values immediately before the service executes. If you specify that you want to validate the outputs of the service, the validation engine validates the service output values immediately after the service executes. An input or output value is invalid if it does not conform to the constraints applied to the input or output parameter.

For input/output validation, the service's declared input and output parameters act as the blueprint or model against which input/output values are validated. To effectively use the input and output parameters as the blueprint for validation, you need to apply constraints to the parameters. For information about applying constraints to variables, see “Applying Constraints to Variables” on page 211. For information about declaring service input and output parameters, see “Declaring Input and Output Parameters for a Service” on page 81.

**Note:** The declared input and output parameters for a service are sometimes called the signature of the service.

You can specify that you want to perform input/output validation for a service in the following ways:

- **Input/Output tab.** Set properties on the Input/Output tab to instruct the validation engine in SAP BC Server to validate the inputs and/or outputs of the service every time the service executes. If a client calls the service and the inputs are invalid, the service fails and does not execute.

- **INVOKE step properties.** Set up input/output validation via the INVOKE step properties to instruct the validation engine to validate the service input and/or output only when it is called from within another flow service. At run time, if the inputs and/or outputs of the service are invalid, the INVOKE flow step that calls the service fails.

To determine which method to use, determine whether or not you want the service input and output values validated every time the service runs. If you want to validate the input and output values every time the service runs, specify validation via the Input/Output tab. For example, if your service requires certain input to exist or fall within a specified range of values, you might want the pipeline validated every time the service runs.

If the input and/or output values do not need to be validated every time the service executes, set up validation via the INVOKE step properties. Specifying input/output validation via the INVOKE step properties allows you to decide on a case-by-case basis whether you want validation performed.
Performing Input/Output Validation

Specifying Input/Output Validation via the Input/Output Tab

You can specify that you want the inputs and/or outputs of a service validated every time it executes by setting properties on the Input/Output tab of the service. Every time the service executes, the validation engine validates the input and/or output values of the service against the input and output parameters declared for the service.

To set up input and output validation via the Input/Output tab

1. In the Service Browser, select the service for which you want to validate input and/or output every time the service is invoked.
2. Click the Input/Output tab.
3. If you want the input of the service validated every time the service executes, select the Validate input check box.
4. If you want the output of the service validated every time the service executes, select the Validate output check box.
5. Click on the Service Browser toolbar to save your changes.

Note: If you specify input/output validation via the INVOKE step, and an input or output value is invalid, the service itself does not actually fail. The validation engine validates input values before SAP BC Server executes the service. If the service input is not valid, the INVOKE flow step for the service fails. Similarly, the validation engine validates output values after SAP BC Server executes the service. If the service output is not valid, the INVOKE flow step for the service fails. Whether or not the entire flow service fails when an individual flow step fails depends on the exit conditions for the service. For information about specifying exit conditions, see “Using SEQUENCE to Specify an Exit Condition” on page 125.
Specifying Input/Output Validation via the INVOKE Step

You can also specify input/output validation by setting properties for the INVOKE step that calls the service. Each time you insert a service into a flow, you can decide whether you want the validation engine to validate service inputs and/or outputs at run time.

To set up input and output validation via the INVOKE step

1. In the Flow Editor, select the service for which you want SAP BC Server to validate input and/or output values.
2. Click the Properties tab.
3. If you want to validate input to the service, in the validate-in list, select $default.
4. If you want to validate the output of the service, in the validate-out list, select $default.
5. Click on the Service Browser toolbar to save your changes
Performing Record and Node Validation

Sometimes, you might want to validate an individual record or node variable in the pipeline instead of the entire pipeline. Using the pub.schema:validate service, you can validate an individual record or node (Document) in the pipeline.

For example, suppose that you invoke the pub.ldap:lookup service in a flow to retrieve an IData object from an LDAP directory service. If you want to validate that object before you use it in other services, invoke the pub.schema:validate service after retrieving the object. As another example, you might want to validate an XML document that has been converted to record. You would use the pub.schema:validate service to validate the record (XML document) against another record.

The pub.schema:validate service considers a record or node to be valid when it complies with the structural and content constraints described in the record or SAP BC schema it is
validated against. This service returns a string that indicates whether validation was successful and an IData array that contains any validation errors. When you insert the pub.schema:validate service into a flow service, you can specify the maximum number of errors to be collected by the service. You can also specify whether the pub.schema:validate service should fail if the record or node variable is invalid.

To validate a record or node in the pipeline

1. In the Service Browser, select the flow service in which you want to validate a Record or node.
2. Click Browse and select Browse.
3. Use the list next to Folder to navigate to and select the pub.schema:validate service.
4. In the Flow Editor, move the validatePipeline service to the point in the flow where you want to validate the record or node.
5. Click the Pipeline tab.
6. Map the variable that you want validated in Pipeline In to object in Service In.
7. Under Service In, select conformsTo and click .
8. In the Input for dialog box, do one of the following:
   
   — If you are validating a record variable, type the fully qualified name of the record that you want to validate the variable against. For example: folderName:recordName. Click OK.
   
   — If you are validating a node (Document), type the fully qualified name of the SAP BC schema that you want to validate the node variable against. For example: folderName:schemaName. Click OK.

Note: The nominated SAP BC schema is only used to validate unqualified nodes, i.e., nodes with XML Namespaces URI = null.
Performing Record and Node Validation

Assign values to the remaining **Service In** variables, using the following table. All of these variables are optional.

<table>
<thead>
<tr>
<th><strong>For this property...</strong></th>
<th><strong>Specify...</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>maxErrors</strong></td>
<td>Number of errors to be collected. When the number of errors found is equal to <em>maxErrors</em>, the validation engine stops validating the record or <em>node</em> and returns the result. If <em>maxErrors</em> is set to -1, the server returns all of the errors. Default is 1.</td>
</tr>
<tr>
<td><strong>ignoreContent</strong></td>
<td>Whether the validation engine should ignore content (values of variables).</td>
</tr>
<tr>
<td>True</td>
<td>Instructs the validation engine to ignore the contents of variables. The validation engine validates the record or <em>node</em> for structural constraints only.</td>
</tr>
<tr>
<td>False</td>
<td>Specifies that the validation engine does not ignore the contents of variables. The validation engine validates the record or <em>node</em> for content and structural constraints. Default is false.</td>
</tr>
<tr>
<td><strong>failIfInvalid</strong></td>
<td>Whether the pub.schema:validate service fails if the record or <em>node</em> is invalid:</td>
</tr>
<tr>
<td>True</td>
<td>Fails the service if the record or <em>node</em> is invalid. When the service fails, an exception is thrown and the errors are embedded in the exception message.</td>
</tr>
<tr>
<td>False</td>
<td>Indicates service success even if the record or <em>node</em> is invalid and returns errors via the pipeline. Default is false.</td>
</tr>
</tbody>
</table>

Click on the Service Browser toolbar to save your changes to the flow service.
Validating a Record in the Pipeline

For more information about the validation built-in services, see the SAP BC Built-In Services Guide.

**Note:** If you want to validate String, String List, or String Table variables in the pipeline, use the pub.schema:validatePipeline. Define a record that contains the variables you want to validate and apply constraints to the variables. Then use this record as the blueprint for the pub.schema:validatePipeline service. Only the variables in the record will be validated. The validation engine ignores other variables that exist in the pipeline at run time. (The record implicitly allows unspecified variables to exist at run time.)
Validation Errors and Exceptions

During data validation, the validation engine generates errors when it encounters values that do not conform to the structural and content constraints specified in the blueprint. The format in which the validation engine returns errors depends on whether validation was performed using the built-in services or by checking the declared input and output parameters for the service.

- When the validation engine performs data validation by executing the built-in services pub.schema:validate or pub.schema:validatePipeline, errors are returned in the `errors` output variable (an IData list). For each validation error, the `errors` variable lists the error code, the error message, and the location of the error.

- When the validation engine performs validation by comparing run-time data to the declared input and output parameters, the validation engine returns all the validation errors in a string. This string contains the error code, error message, and error location for each error found during input/output validation.

For more information about validation errors, see Appendix H, “Validation Errors and Exceptions” on page 627.

Validation Exceptions

If you use the pub.schema:validate and pub.schema:validatePipeline services to perform data validation, you can determine whether the service should succeed or fail if the data being validated is invalid. You might want to a service to succeed even if the data is invalid. In the pub.schema:validate and pub.schema:validatePipeline services, the value of the `failIfInvalid` input variable determines whether a service fails because of an invalid object.

If the pub.schema:validate and pub.schema:validatePipeline service fails, SAP BC Server throws a validation exception. A validation exception is generated if one of the following is true:

- Errors are detected in the object (XML document, pipeline, or Record) that is passed (e.g., null value)

- The basic validation contract is violated (e.g., a binary tree is passed instead of a record as expected)

- You specify that the service should fail if the object to be validated (XML document, pipeline, or record) did not conform to the SAP BC schema or record (e.g., `failIfInvalid` = true). If this is the reason for the exception, SAP BC Server inserts the validation errors into the exception message.
Running Out of Memory During Validation

During validation of an XML document, a large maxOccurs value for an element in the SAP BC schema used as the blueprint can cause an out of memory error or a stack overflow. To prevent a stack overflow or out of memory error, you can set a threshold value for maxOccurs. When the validation engine encounters a maxOccurs value greater than the threshold value, it proceeds as if the maxOccurs value was equal to 'unbounded'.

To set a maxOccurs threshold value, you can edit the server configuration parameter `watt.core.schema.maxOccursThresholdValue`. By default, this parameter does not have a value.

To set a maxOccurs threshold value

1. Start SAP BC Server and open the Server Administrator.
2. In the Settings menu of the navigation area, click Extended.
3. Click Edit Extended Settings. The server displays the Extended Settings screen.
4. In the text area under Extended Settings, type `watt.core.schema.maxOccursThresholdValue=value` where `value` is the number you want to use as the maxOccurs threshold.
5. Click Save Changes.
Managing Service and Record Dependencies

- Basic Concepts ................................................. 236
- Finding SAP BC Elements in the Service Browser ......................... 238
- Finding Dependents .................................................. 240
- Checking Dependencies when Renaming SAP BC Elements ............. 242
- Finding References .................................................. 242
- Inspecting Pipeline References ........................................ 244
Basic Concepts

In SAP BC Developer, you can resolve relationships between SAP BC elements as part of your debugging and collaborative development processes. For example, before you rename a record, you may want to make sure that you do not destroy any references to that record in any other services, records, or specifications.

Before you use the procedures in this chapter, you may want to familiarize yourself with the following concepts.

What Is an SAP BC Element?

An SAP BC element is an item that exists in the Service Browser (left-hand area) in SAP BC Developer. SAP BC elements include services, specifications, records, and SAP BC schemas. Packages and folders are not SAP BC elements.

What Is a Dependent?

In Developer, a dependent is an SAP BC element that uses a selected SAP BC element. For example, suppose that the flow service ServiceA invokes the built-in service pub.web:loadDocument. The ServiceA uses the pub.web:loadDocument service; therefore, flow ServiceA is a dependent of pub.web:loadDocument.
Basic Concepts

Dependent SAP BC Elements

What Is a Reference?

In Developer, a reference is an SAP BC element that is used by a selected SAP BC element. For example, suppose that the flow service ServiceA invokes the services pub.web:loadDocument, ProcessPO and SubmitPO, and declares a Record Reference to PORecord as part of its input signature. The services pub.web:loadDocument, ProcessPO and SubmitPO, and the record reference PORecord are used by ServiceA; therefore, they are references of ServiceA.

What Is a Pipeline Reference?

A pipeline reference is a transformer or pipeline modifier (Map, Drop, or Set Value) involving a variable in a record reference. For example, if ServiceA declares a Record Reference to PORecord as a variable in its input signature, and ServiceA maps a variable from PORecord to a new variable in the pipeline, then that mapping is a pipeline reference.
Finding SAP BC Elements in the Service Browser

When developing services, you might lose track of where you saved certain SAP BC elements. For example, suppose that you do not remember the folder to which you saved a service called “Test.” Using the Find in Service Browser command, you can specify a name and search across all packages and folders to find the “Test” service.

The Find in Service Browser command interprets search terms as regular expressions. By default, the command looks for all SAP BC elements containing a specified search term. For example, if you specified “Test” as a search term, the results would include SAP BC elements named “Test,” “MyTest,” and “TestFinal.”

Note that this functionality searches the fully qualified names of SAP BC elements. That is, if you search for the name “Test,” the results display all elements with “Test” in their fully qualified name. This includes a service called Sample located in a Test folder, or a record called SampleTest.

**Note:** The Find in Service Browser command supports regular expressions but not conditional statements. For example, you can use:

Test

but not

Test OR Test1
To find an SAP BC element in the Service Browser

1. On the Edit menu, click Find, and click In Service Browser.

2. In the Find In Service Browser dialog box, type any portion of the fully qualified name of the SAP BC element that you want to find. Keep in mind that the search term is interpreted as a regular expression and is case sensitive. For example:

<table>
<thead>
<tr>
<th>To find this...</th>
<th>Type this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>All SAP BC elements containing “PO”</td>
<td>PO</td>
</tr>
<tr>
<td>All SAP BC elements starting with “PO”</td>
<td>^PO</td>
</tr>
<tr>
<td>All SAP BC elements ending with “PO”</td>
<td>PO$</td>
</tr>
<tr>
<td>All services with the exact name of “logPO”</td>
<td>:logPO$</td>
</tr>
<tr>
<td>All SAP BC elements containing “log” followed by any two characters (wildcards)</td>
<td>log..</td>
</tr>
</tbody>
</table>

Find in Service Browser Dialog Box

3. If you want to limit the scope of the search to a specific package, select the package in the Package list.

4. Click Find. The Find In Service Browser dialog box displays the results of the search.
Results of Search for "log"

The term "log" is found in...

...the names of 17 elements in the Service Browser. All of these elements contain 'log' in their fully qualified name.

5 To jump to an element in the Service Browser, select that element in the results and click Go To.

Finding Dependents

In Developer, a dependent is an SAP BC element that uses a selected SAP BC element. For example, suppose that the flow service ServiceA invokes the flow service Purchasing:SubmitPO. The ServiceA uses the Purchasing:SubmitPO service; therefore, ServiceA is a dependent of Purchasing:SubmitPO. If you delete Purchasing:SubmitPO from the Service Browser, ServiceA will not run.

During debugging, you might want to locate all of the dependents of a given service or record. Use the Find Dependents command to find all the dependents.

The following are not considered dependents:

- A variable with a content type constraint that references a simple type in an SAP BC schema. If, for example, an SAP BC schema contains the simple type sku and you select sku as the content type constraint for a variable, the SAP BC schema is not a dependent of the variable (or vice versa).

- Java or WebTap services that invoke other services. For example, if Java service A invokes service B, then A will not appear as a dependent of B.
To find dependents of a selected SAP BC element

1. In the Service Browser, select the element for which you want to find dependents.

2. On the Edit menu, click Find, and then click Dependents.

The Find Dependents dialog box displays the dependents of the SAP BC element. The results do not include SAP BC schemas or Java services.

3. After Developer finds the dependents of the selected SAP BC element, you may do any of the following:

   — To jump to an element in the Service Browser, select that element in the results, and click GoTo.

   — To see all child dependents of a found dependent, click next to the item in the results list.

   — To limit the scope of the search to a specific package, select the package in the Package list. Click Find.
Checking Dependencies when Renaming SAP BC Elements

You can instruct Developer to automatically check for dependents when you rename an SAP BC element in the Service Browser. If elements are found that use the element to be renamed, then those dependents are listed. You are given the opportunity to do one of the following:

- Rename the selected SAP BC element and all references in dependent flow services, records, and specifications.
- Rename the selected SAP BC element only.
- Cancel the operation.

To check for dependencies when renaming elements

1. On the Edit menu, click Preferences.
2. On the General tab, under Service Browser, select the Check dependency when renaming/moving check box.
3. Click OK.

For more information about enabling safeguards for renaming, deleting, or moving an element, see “Renaming and Deleting Elements” on page 37.

**Important!** You cannot use the Undo command after renaming an item.

Finding References

In Developer, a reference is an SAP BC element that is used by a selected SAP BC element. For example, suppose that the flow service ServiceA invokes the flow services ProcessPO and SubmitPO, and declares a Record Reference to PORecord as part of its input signature. The flow services ProcessPO and SubmitPO, and record reference PORecord are used by ServiceA; therefore, they are references of ServiceA.

During debugging of a complex flow service, you might want to locate all of the services, records, and specifications used by the flow service. Use the Find References command to locate the references.
The following are not considered references:

- A variable with a content type constraint that references a simple type in an SAP BC schema. If, for example, an SAP BC schema contains the simple type `sku` and you select `sku` as the content type constraint for a variable, the SAP BC schema is not a reference of the variable (or vice versa).

- Java or WebTap service invocations. For example, if Java service A invokes service B, then B will not appear as a reference of A.

**Note:** When you find references, you search across all packages on SAP BC Server. You cannot limit your search to a specific package.

### To find references of a selected SAP BC element

1. In the Service Browser, select the element for which you want to find references.
2. On the Edit menu, click Find, and then click References.

The Find References dialog box displays the references of the SAP BC element. The results do not include Java services, C services, or SAP BC schemas.

Unresolved references are indicated in *bold italics*. An unresolved reference is an element that does not exist in the Service Browser, yet it is still referred to in the service, record, or specification that you selected. The element might have been renamed, moved, or deleted. To prevent unresolved references, enable the renaming and deleting safeguards on the General tab of the Preferences dialog box. For more information, see “Renaming and Deleting Elements” on page 37.
3 After Developer finds the references of the selected SAP BC element, you may do any of the following:
   - To jump to an element in the Service Browser, select that element in the results, and click Go To.
   - To see all child references of a found reference, click next to the item in the results list.

Inspecting Pipeline References

A pipeline reference is a transformer or pipeline modifier (Map, Drop, or Set Value) involving a variable in a record reference. For example, if ServiceA declares a Record Reference to PORecord as a variable in its input signature, and ServiceA maps a variable from PORecord to a new variable in the pipeline, then that mapping is a pipeline reference. When editing a complex record, you might want to check all dependent pipeline modifiers for validity. Use the Inspect Pipeline References command to locate and inspect pipeline references.

When inspecting pipeline references, keep the following points in mind:

- You can only inspect the pipeline references of a flow service or record reference. The search results include only flow services or record references that contain invalid pipeline modifiers or transformers.
- Values set at the top level of a record reference in the pipeline are not considered pipeline references. Therefore, if the top level of a record reference contains input values for a nonexistent field, it will not appear in the search results, although it is invalid.
- The search results will not show variable type and dimensionality mismatches. For example, suppose that you map to a String variable named PONum within record reference PORecord. However, variable PONum in PORecord is actually a String List. This dimensionality mismatch will not appear in the search results.
- If you are inspecting the pipeline references of a flow service, you can only inspect references across all packages on SAP BC Server.
- If you are inspecting pipeline references of a record reference, you can inspect references across a specific package or all packages.
To inspect pipeline references

1. In the Service Browser, select the flow service or record for which you want to find invalid pipeline references.

2. On the Edit menu, click Inspect Pipeline References.

   The Inspect Pipeline References dialog box displays all invalid pipeline references for the selected service or record.
   - If you inspected a flow service, the search results contain all of the records that have invalid pipeline references in that flow.
   - If you inspected a record, the search results contain all of the flow services that have invalid pipeline references to that record.

   The PO:processPO flow service contains... ...a reference in its pipeline to the nonexistent record PO:POstructure.

3. After Developer finds the pipeline references of the selected SAP BC element, you may do any of the following:
   - To jump to an element in the Service Browser, select that element in the results, and click GoTo.
   - To jump to the unresolved reference in the pipeline, select the element in the results, and click Find in Flow.
   - If the selected element has multiple unresolved references in the same flow service, you can use the Find Next command to automatically jump to the next reference within the selected element. To use the Find Next command, keep the Inspect Pipeline References dialog box open. Then select Edit ▶ Find Next or press CTRL+F3.
— If the selected element is a record reference and you want to limit the scope of the search to a specific package, select the package in the Package list. Click Inspect.
CHAPTER 11

Testing and Debugging Services

- Testing and Debugging ................................................................. 248
- Testing Services ................................................................. 248
- Testing Services from Developer ........................................... 249
- Testing Services from a Browser ........................................... 257
- Testing Services that Expect XML Documents as Input ............... 258
- Working in Debug Mode ......................................................... 259
- Setting Breakpoints ............................................................... 266
- Disabling Flow Steps, Transformers, and Conditions ..................... 269
- Modifying the Current Pipeline ............................................... 274
- Saving and Restoring the Pipeline ........................................... 275
- Other Debugging Techniques .................................................. 280
Testing and Debugging

Like other types of computer programming, developing a service is an iterative process of building, testing, and correcting (debugging) your code.

Developer provides a range of tools to assist you during the testing and debugging phases. For example, you can:

- Test services, specify their input values, and inspect their results.
- Examine the call stack and the pipeline when an error occurs.
- Execute services in “debug” mode, a mode that lets you monitor a flow service’s execution path and/or execute its steps one at a time.
- Temporarily disable steps in a flow and/or specify points where you want to halt execution.

Additionally, SAP provides tools for collecting run-time information that can help debug a service. These include tools to:

- Write arbitrary messages to the server log.
- Trace the pipeline at run time.
- Modify and save the contents of the pipeline at a specified point.

Testing Services

You can test any type of service—flow services or coded services—with Developer, eliminating the need to build special clients simply for testing. It also allows you to easily pass different sets of test data to your service, which makes it easy to test your service under a variety of data conditions.

You can use Developer to test services in two ways:

- **From Developer.** With this technique, Developer is the client—that is, it invokes the service and receives the results.
  
  -OR-

- **From a browser.** With this technique, Developer formulates the URL necessary to invoke the service and passes that URL to your browser. Your browser actually invokes the service and receives the results. When you develop services for browser-based clients—especially ones whose output will be formatted using output templates—you will want to test those services at least once using this technique.
Testing Services from Developer

In most cases, you will test your services using Developer (not a browser) as the client. You do this using the Run command on the Test menu.

When you execute a service with the Run command, Developer invokes the service (just as an ordinary SAP BC client would) and receives its results. The service executes once, from beginning to end (or until an error condition forces it to stop) on the server on which you have an open session.

Before Developer invokes the service, it prompts you for input values. You can type the input values into the dialog box provided by Developer or load the values from a file that was saved during an earlier test.

Results from the service are returned to Developer and displayed on the Results tab. This allows you to quickly examine the data produced by the service and optionally change it or save it to a file. You can use the saved data as input for a later test or to populate the pipeline during a debugging session.

To test a service using Developer as the client

1. In the Service Browser, select the service that you want to run.
2. On the Test menu, select Run. If you have not yet saved changes to the service, Developer prompts you to save them.
3. If the service has input parameters, type the input values for each variable in the Input dialog box or click the Load button to retrieve the values from a file. If you need procedures for this step, see “Entering Input for a Service” on page 250.
4. If you want Developer to pass empty variables (variables that have no value) to the service, select the Include Empty Values check box. When you select this option, empty Strings are passed with a zero-length value. If you do not select this option, empty Strings are not passed to the service.
5. If you want to save the input values that you have entered, click Save. Input values that you save can be recalled and reused in later tests. For additional information about saving input values, see “Saving Input Values to a File” on page 251.
6. Click OK. Developer invokes the service with the specified set of input values.

   - If the service executes to completion, its results appear on the Results tab. For more information about the Results tab, see “Viewing the Results of the Service” on page 253.
   - If the service throws an exception, an error message displays. Click Details in the message box to view the call stack and the state of the pipeline where the error occurred. For additional information about errors that occur while testing a service, see “Run-Time Exceptions” on page 255.
CHAPTER 11 Testing and Debugging Services

Entering Input for a Service

When you test a service with Developer, Developer opens the Input dialog box, which prompts you for input to the service. This dialog box contains an entry for each string-based variable defined by the service’s input parameters. String-based variables are Strings, String Lists, and String Tables.

Records and Record Lists containing string-based children can also be entered through the Input dialog box.

Enter input values in the Input dialog box

You can enter simple Strings...

...and complex Records and Record Lists.

You can manually type your input values into the Input dialog box or, if you saved the input values from an earlier test, you can load them from a file.

Note: If your service takes variables that are not string-based (Objects), you will not be able to enter those values in the Input dialog box (such values do not even appear in the Input dialog box). To test this type of service, you must also create a service that generates input values for your service. Then you need to construct a test harness—a flow service that executes both the service that produces the test data and the service you want to test—and use that harness to test your service.
To enter values by typing them

1. Open the service and execute it as described in “Testing Services from Developer” on page 249 or “Testing Services from a Browser” on page 257.

2. For each variable listed in the Input dialog box, type an input value. If the service takes complex variables such as a String Lists, Records, or Record Lists, use the following buttons to specify the variable's individual elements.

   **Use this...**  
   **To...**
   ![Add a row to the variable.](image)
   ![Insert a blank row above the currently selected row.](image)
   ![Add a column to the variable.](image)
   ![Delete the selected row from the variable.](image)
   ![Delete the selected column from the variable.](image)

3. If you want Developer to pass empty Strings to the service, select the **Include Empty Values** check box. When you select this check box, empty Strings are passed with a zero-length value. If you do not select this check box, empty strings are not passed to the service.

**Saving Input Values to a File**

You can save values that you type in the Input dialog box to a file so that you can reuse them in later tests.

When saving input values to a file, keep the following points in mind:

- Empty variables (variables that do not have a value) are only saved if the **Include Empty Values** option is selected. If you do not select this option, Developer does not save empty variables in the file.

- You can store the file in any directory that is accessible to the computer on which Developer is running. Because these files are not actual run-time components, they do not need to be saved in SAP BC Server's namespace or even on the server machine itself.
Note: You might want to consider creating an input file for each set of data that you routinely test your service against. This will provide you with a ready-made set of test cases against which to verify the service when it is modified by you or other developers in the future. Many sites establish a special directory on their development server just for holding sets of test data that they generate in this manner.

To save input values that you have entered for a service

1. Open the service and execute it as described in “Testing Services from Developer” on page 249 or “Testing Services from a Browser” on page 257.
2. Enter input values into the Input dialog box as described in “Entering Input for a Service” on page 250.
3. When you are finished entering values, click Save.
4. In the Save File dialog box, specify the name of the file to which you want the values saved.

Loading Input Values from a File

You can reload input values that you have saved to a file instead of re-keying the values in the Input dialog box. To do this, click the Load button in the Input dialog box and then select the file that contains the input values you want to use.

When you use input values from a file, keep the following points in mind:

- Developer only loads variables whose name and type match those displayed in the Input dialog box. Variables that exist in the file, but not in the dialog box, are ignored.
- Values from the file replace those already in the Input dialog box.
- Variables that exist in the Input dialog box, but not in the file are set to null.
- Besides loading values that were saved from the Input dialog box, you can also load values that were saved using pub.flow:savePipelineToFile or the Save Pipeline commands on the Results tab. In addition, you can change values in the pipeline during testing. For information about saving the state of the pipeline, see “Saving and Restoring the Pipeline” on page 275. For information about modifying values in the pipeline, see “Modifying the Current Pipeline” on page 274.
To load input values that have been saved to a file

1. Open the service and execute it as described in “Testing Services from Developer” on page 249.

2. When the Input dialog box appears, click Load. The Load File dialog box appears.

3. Select the file containing the input values that you want to load.

**Viewing the Results of the Service**

When you execute a service with the Run command or with one of Developer's debugging tools, its results are displayed on the Results tab.

**Results from a service you run in Developer are displayed on the Results tab**

The upper half of the Results tab displays all the variables in the pipeline. To view the contents of a particular variable, you select the variable in the upper half. Its contents are shown in the lower half.
When viewing the **Results** tab, keep the following points in mind:

- The **Results** tab represents the contents of the pipeline.
- The **Results** tab shows *all* variables placed in the pipeline by the service, not just those that were declared in the service's input/output parameters.
- Variables that a service explicitly drops from the pipeline do not appear on the **Results** tab.
- You can browse the contents of the **Results** tab, but you cannot edit it directly.
- You can save the contents of the **Results** tab to a file and use that file to restore the pipeline at a later point. For additional information about saving and restoring the contents of the **Results** tab, see “Saving and Restoring the Pipeline” on page 275.
- If you run a service and an error occurs, results are not displayed in the **Results** tab. However, you can click the **Details** button in the error message box to examine the state of the pipeline at the point where the exception occurred.
- When debugging a flow service in step mode, you can use the **Results** tab to examine the state of the pipeline after each flow step. You can optionally load a different pipeline or modify the existing pipeline between steps. For information about loading a pipeline in the **Results** tab, see “Saving and Restoring the Pipeline” on page 275. For information about modifying an existing pipeline, see “Modifying the Current Pipeline” on page 274.
- When you use a breakpoint or the **Trace to Here** command to halt execution of a flow service, you can use the **Results** tab to examine the pipeline at that point where you halted the flow. You may also optionally load a different pipeline or modify the existing pipeline at this point. For information about loading a pipeline in the **Results** tab, see “Saving and Restoring the Pipeline” on page 275. For information about modifying an existing pipeline, see “Modifying the Current Pipeline” on page 274.
- Variables whose object types are not directly supported by the Developer will appear in the **Results** tab, but because Developer cannot render the values of such objects, a value does not appear in the **Values** column. Instead, the **Values** column displays the object’s Java class message.
- Variables that contain `com.wm.util.Table` objects appear as Record Lists in the **Results** tab.
Copying Variables From the Results Tab

You can use the Copy command to copy information from the Results tab and paste it into other fields in Developer. The information that Developer inserts when you paste an element from the Results tab depends on the field into which you paste it.

If you paste to a field that expects...

A data definition (e.g., the Pipeline Editor, the Input/Output tab, or a Record).
The name of an element.

Developer inserts...

The copied element’s data definition.
The copied element’s name (and position if it is a member of a complex element such as a String Table, Record, or Record List).

To copy and paste elements from the Results tab

1. Display the Results tab and select the element that you want to copy. (When copying elements from the lower half, you can select a group of contiguous elements by pressing the SHIFT key and selecting the first and last element in the group that you want to copy.)

2. Select the Copy command from the Edit menu or right-click and select Copy.

3. Select the field into which you want to paste the information, then select the Paste command from the Edit menu or right-click and select Paste. (If the Paste command is not available, it indicates that the information cannot be pasted into the selected field.)

Run-Time Exceptions

If a service that you run from Developer throws an exception, Developer reports the error in the following message box:

An error message appears when a service fails

You can click the Details button to display the call stack and the pipeline at the point where the error occurred.
The Details button shows the Call Stack and the Pipeline

The Call Stack

The call stack identifies which flow step generated the error and lists its antecedents. For example, let’s say you have a service called PARENT that invokes three services, CHILD_A, CHILD_B, and CHILD_C. If CHILD_B is a Java service and it throws an exception, the call stack will look like this:

Call Stack from a nested service

Note: You can improve performance and memory usage in Developer by caching SAP BC elements, such as services and schemas. For details, see “Caching Elements” on page 50.
Now let’s assume that CHILD_B is a flow service that calls three Java services: CHILD_B1, CHILD_B2, and CHILD_B3. If CHILD_B3 throws an exception, the call stack will look like this:

**Call Stack from a deeply nested Service**

<table>
<thead>
<tr>
<th>Service</th>
<th>Flow Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOXOrders.CHILD_B3</td>
<td>Invoke CHILD_B3</td>
</tr>
<tr>
<td>SOXOrders.CHILD_B</td>
<td>Invoke CHILD_B2</td>
</tr>
<tr>
<td>SOXOrders.PARENT</td>
<td>Invoke CHILD_B</td>
</tr>
</tbody>
</table>

Note that the call stack is LIFO-based. That is, the top entry in the stack identifies the last (i.e., most recent) service invoked. The bottom entry identifies the parent service—the one that you originally invoked from Developer. If the parent itself throws the exception, it will be the only entry in the call stack.

**Note:** Services invoked from within a Java service are not reported in the call stack, even if they throw the exception.

### The Pipeline Dump

The Server Failures Detail dialog box contains the contents of the pipeline at the point of failure. Note that when a failure occurs within a Java service, the Pipeline area represents the state of the pipeline at the point when that service was initially called. If the Java service made changes to these values before throwing the exception, those changes will not be reflected in the Pipeline information.

### Testing Services from a Browser

You can use the Run in Browser command in Developer to test a service from a browser (i.e. to simulate a browser-based client). When you use this command, Developer prompts you for the service’s input values, builds the URL necessary to invoke the service with the inputs you specify, and then passes the URL to your browser. When you use this command to test a service, your browser (not Developer) actually invokes the service and receives its results.
If you are developing services that will be invoked by browser-based clients, particularly ones whose output will be formatted using output templates, you will want to test those services using Run in Browser command to verify that they work as expected.

To test a service using a browser as the client

1. In the Service Browser, select the service that you want to run.
2. From the Test menu, select Run in Browser. If you have unsaved edits, Developer prompts you to save them.
3. If the service has input parameters, type the input values for each variable in the Input dialog box or click the Load button to retrieve the values from a file. If you need procedures for this step, see “Entering Input for a Service” on page 250.

   **Note:** Only Strings and String Lists are passed to the browser.

4. If you want to pass empty variables (variables that have no value) to the service, select the Include Empty Values check box. When you select this option, empty Strings are passed with a zero-length value. If you do not select this option, Developer excludes empty variables from the query string that it passes to the browser.
5. If you want to save the input values that you have entered, click Save. Input values that you save can be recalled and reused in later tests. For additional information about saving input values, see “Saving Input Values to a File” on page 251.
6. Click OK. Developer builds the URL to invoke the service with the inputs you have specified, launches your browser, and passes it the URL.
   - If the service executes successfully, its results appear in your browser. (If an output template is assigned to the service, the template will be applied to the results before they are returned.)
   - If the service experiences an error, an error message is displayed in the browser.

Testing Services that Expect XML Documents as Input

If your service expects an XML document as input (i.e., it takes a node as input), you can test it using the Send XML File command. This command prompts you for the name of the XML file and submits the file to the service, emulating the way in which the service would execute if an XML document were posted to it.
To test a service that expects an XML document as input

1. In the Service Browser, select the service that you want to run.

2. From the Test menu, select Send XML File. If you have unsaved edits, Developer prompts you to save them.

3. In the Select Test Mode dialog box, specify whether you want the service to run in Trace mode or Step mode and click OK.

4. In the Open File dialog box, select the XML file that you want to submit to this service and click OK. Developer submits the file to the server, which parses it into a node object and passes it to selected service.

   — If the service executes to completion, its results appear on the Results tab. For more information about the Results tab, see “Viewing the Results of the Service” on page 253.

   — If the service experiences an error, an error message displays. Click Details in the message box to view the call stack and the state of the pipeline where the error occurred. For additional information about errors that occur while testing a service, see “Run-Time Exceptions” on page 255.

Working in Debug Mode

When you use Developer to execute a service using the Run or Run In Browser commands, it executes as a normal service. That is, the server does not execute it any differently than it would if any other client requested it. However, Developer also allows you to test a service in debug mode, a mode that allows you to monitor the execution path of a flow service and/or move through its steps one at a time.

When you run a flow service in debug mode, it is executed in a special way that returns results and other debugging information back Developer after each step.
CHAPTER 11 Testing and Debugging Services

Entering Debug Mode

You automatically enter debug mode when you select any of the following commands from the Test menu:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace</td>
<td>Executes flow steps one after another to the end of the service and visually marks steps as they execute. For information about using this command, see “Using the Trace Tools” on page 262.</td>
</tr>
<tr>
<td>Trace to Here</td>
<td>Executes flow steps one after another up to a specified point and visually marks steps as they execute. For information about using this command, see “Using the Trace Tools” on page 262.</td>
</tr>
<tr>
<td>Trace Into</td>
<td>Executes flow steps one after another to the end of the service and visually marks steps as they execute, including steps in child flows. For information about using this command, see “Tracing into a Child Flow” on page 263.</td>
</tr>
<tr>
<td>Step</td>
<td>Executes the next flow step and then halts. For information about using this command, see “Using the Step Tools” on page 264.</td>
</tr>
<tr>
<td>Step Into</td>
<td>Opens a child flow or a MAP step so that you can debug the individual flow steps within it. For information about using this command, see “Stepping though a Child Flow” on page 265 and “Using the Step Tools with a MAP Step” on page 266.</td>
</tr>
</tbody>
</table>

⚠️ **Important!** The debug commands are only available for flow services. When a Java service is selected, these commands are not available.

When you first enter debug mode, processing always starts from the first step in the flow service. Completed steps are marked with a gray outline. A step that is “in process” or is the next in line to be processed is marked with a green outline. When you step though a flow service or you halt execution using a breakpoint or the Trace to Here command, the green outline indicates which step will execute when you resume processing.

The example below shows a flow service that is being executed using the Step command. As you can see, the BRANCH on 'PaymentType' step has three targets. The gray outline shows which path was executed. The green outline indicates that BRANCH on '/AuthorizationCode' is the step that will execute when the next Step command is performed.
Debug Mode visually shows you the services' execution path

Combining the Step and Trace Commands in Debug Mode

Once you enter debugging mode, you can switch among the different debugging commands and examine certain segments of the service more closely than others. For example, you might want to execute the first few steps of a service with the Step command and then simply trace the remainder. Or, you might want to use Trace to Here to execute to a particular point and then execute the remaining flow steps in the service one step at a time.

When you combine techniques, remember that the flow step outlined in green always indicates the current point of execution and marks the next flow step that will execute when you resume processing.

Resetting Debug Mode

The following actions will reset your debugging session.

- Executing the Test ➤ Reset command
- Executing the Run command
- Refreshing Developer's session on the server
- Editing the flow service or any other service
- The service throws an exception
When a session is reset, trace lines are cleared and the point of execution is set back to the top of the flow service.

The following actions do not reset a debugging session:

- Setting breakpoints
- Examining the contents of any of the other tabs associated with the service
- Saving or restoring the contents of the Results tab

**Using the Trace Tools**

If a flow service has a complex path of execution (for example, it contains many branching sequences), it is often useful to simply see which path was taken when the flow executed. The trace tools, Trace, Trace to Here, and Trace Into, allow you to do this. They visually mark the flow steps that are executed within a flow service.

<table>
<thead>
<tr>
<th>Use...</th>
<th>If you want to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace</td>
<td>Trace to the end of the service without tracing child services.</td>
</tr>
<tr>
<td></td>
<td>(Breakpoints are ignored in this mode.)</td>
</tr>
<tr>
<td>Trace to Here</td>
<td>Trace to a specified point in the service without tracing child services.</td>
</tr>
<tr>
<td></td>
<td>(Breakpoints are ignored in this mode.)</td>
</tr>
<tr>
<td>Trace Into</td>
<td>Trace to the end of the service or the next breakpoint, including the paths of all child services that this service invokes.</td>
</tr>
</tbody>
</table>

To trace to the end of a service

1. If the service is not already displayed in the Flow Editor, select it from the Service Browser.
2. On the Test menu, click Trace.
   - If the service is already in debug mode, Developer traces the remaining steps, starting from the current point of execution (the step outlined in green.)
   - If the service is not in debug mode, Developer starts the trace at the top of the flow. If you have any unsaved changes, Developer prompts you to save those changes before it starts the trace. If the service takes input, you will be prompted for its input values.
To trace to a specified point

1. If the service is not already displayed in the Flow Editor, select it from the Service Browser.

2. In the Flow Editor, select the flow step up to which you want to trace. (Developer will trace all steps up to, but not including, the one you select.)

   **Note:** If the point to which you want to trace resides in a child of the service that you are testing, you must use the breakpoint feature to trace to that point. For information about setting breakpoints, see “Setting Breakpoints” on page 266.

3. Select **Trace to Here** from the **Test** menu.
   - If the service is already in debug mode, Developer starts at the current point of execution (the step outlined in green) and traces to the selected step. If the selected step is before the current point of execution, the trace starts at the top of the flow.
   - If the service is not in debug mode, Developer starts the trace at the top of the flow service and traces to the selected step. If you have any unsaved changes, Developer prompts you to save those changes before it starts the trace. If the service takes input, you will be prompted for its input values.

4. When Developer reaches the selected flow step, it halts. At this point, you may do any of the following:
   - Examine the contents of the **Results** tab.
   - Modify, save, and/or restore the contents of the **Results** tab.
   - Use **Step** or **Step Into** to execute subsequent flow steps one at a time.
   - Select another step in the flow service and use **Trace to Here** to trace to that point.
   - Select **Trace** to trace the remainder of the service.
   - Select **Reset** to clear the debugging session and reset the starting execution point to the top of the service.

**Tracing into a Child Flow**

Many times, the service you are debugging invokes other flow services (child services). In these cases it is useful to trace the execution paths of the child services as well as the parent that you are testing. You do this with the **Trace Into** command.

When you initiate a trace with **Trace Into**, Developer automatically opens and traces the individual steps in every child flow that the parent invokes, including the children of the child services if there are any.
To trace into a child service

1. If the parent service you want to test is not already displayed in the Flow Editor, select it from the **Service Browser**.
2. On the **Test** menu, click **Trace Into**.

   - If the service is already in debug mode, Developer starts the trace at the current point of execution (the step outlined in green) and traces the service and its children until it reaches a breakpoint or the end of the flow.
   
   - If the service is not in debug mode, Developer starts the trace at the top of the flow and traces the service and its children until it reaches a breakpoint or the end of the flow. If you have any unsaved changes, Developer prompts you to save those changes before it starts the trace. If the service takes input, you will be prompted for its input values.

**Using the Step Tools**

You use the **Step**, **Step Into**, and **Step Out** commands on the **Test** menu to interactively execute a flow service one flow step at a time. Stepping through a flow is a very effective debugging technique, because it allows you to examine (and optionally modify) the data in the pipeline before and after each step. Additionally, if you are trying to isolate an error, step mode can quickly help you pinpoint the offending flow step.

<table>
<thead>
<tr>
<th>Use...</th>
<th>If you want to...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step</strong></td>
<td>Execute the current flow step (the one with the green outline).</td>
</tr>
<tr>
<td><strong>Step Into</strong></td>
<td>Open a child flow so that you can debug the individual flow steps within it.</td>
</tr>
<tr>
<td><strong>Step Out</strong></td>
<td>Return to the parent flow from a child that you have stepped into.</td>
</tr>
</tbody>
</table>

Note that at any point while stepping through a flow service, you can do any of the following:

- Examine the contents of the **Results** tab.
- Modify, save, and/or restore the contents of the **Results** tab.
- Select a step in the flow service and use **Trace to Here** to trace to that point.
- Select **Trace** to trace the remainder of the service.
- Select **Reset** to clear the debugging session and reset the starting execution point to the top of the service.
To step through a flow service

1. If the service that you want to step through is not already displayed in the Flow Editor, select it from the Service Browser.

2. Select Step from the Test menu.
   - If the service is already in debug mode, Developer executes the current step (the step outlined in green) and then stops.
   - If the service is not in debug mode, Developer enters debug mode and selects the first step in the flow service. To execute that flow step, select Step again. If you have any unsaved changes, Developer prompts you to save those changes before it enters debug mode. If the service takes input, you will be prompted for its input values.

Stepping though a Child Flow

Many times, the flow service you are debugging invokes other flow services (child services). In these cases it is useful to step through the individual flow steps within a child service, too. You do this with the Step Into and Step Out commands.

To step into and out of a child flow

1. Select the parent flow service and step or trace to the flow step that invokes the child flow. See “To step through a flow service” on page 265 or “To trace to a specified point” on page 263 if you need procedures for this step.

2. Select Step Into from the Test menu. Developer opens the child flow service and selects (but does not execute) the first step.

3. Select Step from the Test menu to execute the first step in the child service. Repeat this step for each flow step that you want to individually execute within the child.

4. If you want to return to the parent flow service without stepping through the entire child, select Step Out from the Test menu. This command will trace the remaining steps in the child flow, return to the parent, and then select, but not execute the next step in the parent flow.

Notes:

- While you are debugging the child, you may use Trace to Here or set a breakpoint to execute up to particular point in the child.
- If you select Trace or Trace Into while you are debugging the child, Developer traces the remaining steps in the child and returns to the parent automatically.
- If you select Step on the last step in the child flow service, Developer automatically returns you to the parent.
— You can use Step Into to step into a child flow that is nested within a child that you have stepped into.
— If you select Step Into on a step that is not a flow service, Step is executed.

Using the Step Tools with a MAP Step

You can use the Step Into and Step Out commands to debug individual transformers in a MAP step.

To step into and out of a MAP Step

1. Select the parent service that contains the MAP step and then step or trace to the MAP step. (i.e., make the MAP step the current flow step. It must have the green outline.) See “To step through a flow service” on page 265 or “To trace to a specified point” on page 263 if you need procedures for this step.

2. Select Step Into from the Test menu. Developer drops into the Pipeline Editor and selects (but does not execute) the first transformer in the MAP step.

3. Select Step from the Test menu to execute the first transformer. Repeat this step for each transformer that you want to individually execute within the MAP step.

4. If you want to return to the parent without stepping through the entire MAP, select Step Out from the Test menu. This traces the remaining transformers in the MAP, returns to the parent, and selects, but does not execute the next step in the parent flow.

Notes:
— If you select Trace or Trace Into while you are debugging the MAP, Developer traces the remaining steps in the MAP and returns to the parent automatically.
— If you select Step on the last transformer in the MAP, Developer automatically executes that transformer and returns you to the parent flow.
— You can use Step Into to step into a transformer that is a flow service.
— If you select Step Into on a transformer that is not a flow service, Step is executed.

Setting Breakpoints

A breakpoint is a point in a flow service where you want processing to halt when you execute that flow service with certain debug modes. Breakpoints can help you isolate a section of code or examine data values at a particular point in the execution path. For example, you might want to set a pair of breakpoints before and after a particular segment of a flow so that you can examine the pipeline (the Results tab) before and after that segment executes.
When working with breakpoints, keep the following points in mind:

- Breakpoints are not persistent. They exist only during the life of the Developer session on the current server in which you set them. When you close Developer or refresh the session on the current server, your breakpoints are cleared. (Note that resetting debug mode does not clear your breakpoints.)

- Breakpoints are also local to your Developer session on the current server. Breakpoints that you set on your machine do not affect other developers or users who might be executing or debugging services in which you have set breakpoints.

- Breakpoints are only recognized when you execute a service with the **Trace Into** command from Developer. If you execute a service using any of the other testing or debugging commands, breakpoints are ignored.

- If you are caching services using the **Caching** tab in Preferences, and your flow service has a breakpoint, you cannot clear the cache of the flow service until the breakpoint is removed. For more information about caching, see “Configuring a Service’s Use of Cache” on page 88.

### What Happens When a Breakpoint is Encountered?

When you execute a service that contains a breakpoint or call a child service that contains a breakpoint, the service is executed up to, but not including, the designated breakpoint step. At this point, processing stops and will not resume until you select another one of Developer’s debugging commands. (Remember, if you want Developer to stop at subsequent breakpoints, you must select the **Trace Into** command.)

**To set a breakpoint on a flow step**

1. In the Service Browser, select the flow service in which you want to set a breakpoint.
2. In the Flow Editor, select the step that will function as the breakpoint. (During debugging, processing will halt immediately before this step).
3. From the **Test** menu, select **Set Breakpoint**. The step’s icon turns red, indicating that it is a breakpoint.

**To clear a breakpoint from a flow step**

1. In the Service Browser, select the flow service in which you want to clear a breakpoint.
2. In the Flow Editor, select the breakpoint step.
On the Test menu, select Clear Breakpoint. The step’s icon returns to its normal color, indicating that it is no longer a breakpoint.

–OR–

1 On the Test menu, select Breakpoints to display the current list of breakpoints on the current server.

2 In the list, select the breakpoint that you want to clear.

3 Click Remove.

Setting Breakpoints on Transformers

can set a breakpoint on a transformer in a MAP step. When you execute a service that contains a breakpoint or calls a service that contains a breakpoint on a transformer, the service is executed up to, but not including, the designated breakpoint transformer.

**Note:** Transformers in a MAP step execute in an arbitrary order—you cannot assume an order of execution. Consequently, some of the transformers in the MAP step might execute before Developer reaches the breakpoint—even if the transformers appear below the breakpoint in the Pipeline Editor. Likewise, transformers above the breakpoint might not execute before the breakpoint is encountered. (These will execute when you resume tracing.) Executed transformers have a gray outline.

To set a breakpoint on a transformer

1 In the Service Browser, select the flow service in which you want to set a breakpoint.

2 In the Flow Editor, select the MAP step containing the transformer that will function as the breakpoint.

3 In the Pipeline Editor, select the transformer that will function as the breakpoint. (During debugging, processing will halt immediately before this transformer.)

4 On the Test menu, select Set Breakpoint. The ➔ icon appears next to the transformer name, indicating that it is a breakpoint.

To clear a breakpoint on a transformer

1 In the Service Browser, select the flow service in which you want to clear a breakpoint.

2 In the Flow Editor, select the MAP step that contains the transformer from which you want to clear a breakpoint.

3 In the Pipeline Editor, select the transformer from which you want to clear a breakpoint.
Disabling Flow Steps, Transformers, and Conditions

When testing and debugging services, you can disable flow steps and transformers. You can also disable the condition placed on a map between variables in the Pipeline Editor. The follow sections provide more information about disabling each of these items.

Disabling Flow Steps

You use the Compose ▶ Disable Step command to disable a step in a flow service. Steps that you disable are not executed at run time.

Disabled steps appear dimmed when viewed in Developer. If you disable a parent step (e.g., a LOOP or a BRANCH), its children are automatically disabled, too. If you disable a MAP step, the transformers in the MAP step are automatically disabled, too.

Disabling Flow Steps, Transformers, and Conditions

4. On the Test menu, select Clear Breakpoint. Developer removes the ➔ icon next to the transformer name.

–OR–

1. On the Test menu, select Breakpoints to display the current list of breakpoints on the current server.

2. In the list, select the breakpoint that you want to clear.

3. Click Remove.

Viewing a List of Breakpoints

Use the following procedure to view the list of breakpoints that are currently set in your Developer session. From this list, you can also clear and/or go to specific breakpoints.

To display the list of current breakpoints

1. Click ▶ on the toolbar or select Breakpoints from the Test menu.

2. If you want to go to a specific breakpoint, select it and then click Go to Breakpoint.

3. If you want to clear a breakpoint, select it and then click Remove.

**Note:** Remember, breakpoints are not persistent. They only exist during the Developer session on the current server in which you set them. When you refresh or close your session on the current server, your breakpoints are cleared.
Disabled steps are not executed at run time

Disabling a step is useful in many testing and debugging situations. For example, you might want to disable one or more steps to isolate a particular segment of a flow, similar to the way you might “comment out” a section of source code in a program you are testing.

Be aware that disabling a step sets a persistent attribute that is saved in the flow service. Once you disable a step, it remains disabled until you explicitly re-enable it with Developer.

**Important!** The run-time effect of disabling a step is the same as deleting it. Disabling a key step or forgetting to re-enable a disabled step can break the logic of a service and/or cause the service to fail. Developer allows you to disable any step in a flow service, but it is your responsibility to use this feature carefully.

**To disable a step in a flow service**

1. In the **Service Browser**, select the flow service that you want to edit.
2. In the Flow Editor, select the step that you want to disable.
3. Select **Disable Step** from the **Compose** menu.

   –OR–

   Right-click the step, and select **Disable Step**.

   The step dims, indicating that it is disabled.
To enable a step in a flow service

1. In the **Service Browser**, select the flow service that you want to edit.
2. In the Flow Editor, select the disabled step that you want to re-enable.
3. Select **Enable Step** from the **Compose** menu.

   **–OR–**
   - Right-click the step, and select **Enable Step**.
     
     The step dims, indicating that it is disabled.

**Disabling Transformers**

You can also use the **Compose > Disable Step** command to disable a transformer in a MAP step. Transformers that you disable are not executed at run time. In fact, SAP BC Server does not execute any of the maps between pipeline variables and the variables for a disabled transformer.

Disabled transformers appear dimmed when viewed in Developer.
Disabled transformers are not executed at run time

1. In the Service Browser, select the flow service that you want to edit.
2. In the Flow Editor, select the MAP step containing the transformer that you want to disable.
Disabling Flow Steps, Transformers, and Conditions

3 On the Pipeline tab, select the transformer you want to disable.

4 Select Disable Step from the Compose menu.
   –OR–
   Right-click the step, and select Disable Step.
   The transformer dims, indicating that it is disabled.

To enable a transformer in a MAP step

1 In the Service Browser, select the flow service that you want to edit.

2 In the Flow Editor, select the MAP step containing the disabled transformer that you want to enable.

3 On the Pipeline tab, select the disabled transformer that you want to enable.

4 Select Enable Step from the Compose menu.
   –OR–
   Right-click the step, and select Enable Step.

Disabling a Condition Placed on a Map Between Variables

When you map variables to each other, you can apply a condition to the link that connects the variables. At run time, this condition needs to be true for the value of the source variable to be copied to the target variable. During testing and debugging, you might want to disable or remove the condition from the link to make sure that Developer properly copies data between variables. By disabling the condition, you instruct Developer to ignore the condition placed on the link and automatically execute the map (copy the value from the source variable to the target variable).

Disabling the condition preserves the written expression. When you enable the condition, you do not need to rewrite the expression.

The Pipeline Editor uses a blue link (line) to indicate that properties (such as conditions and array indexes) have been applied to the link between variables. Developer keeps the color of the link blue even when you disable the applied condition to remind you that properties have been set.

To disable a condition placed on a map (link) between variables

1 In the Service Browser, select the flow service that you want to edit.

2 In the Flow Editor, select the INVOKE or MAP step that contains the link with the condition you want to disable.

3 On the Pipeline tab, select the link with the condition that you want to disable.
CHAPTER 11  Testing and Debugging Services

4  Right-click the link that connects the variables, and select Properties.

5  In the Link Properties dialog box, click the General tab, and clear the Copy only if check box. Developer dims the field containing the conditional expression.

6  Click OK.

To enable a condition placed on a map (link) between variables

1  In the Service Browser, select the flow service that you want to edit.

2  In the Flow Editor, select the INVOKE or MAP step containing the link with the condition you want to enable.

3  On the Pipeline tab, select the link with the condition that you want to enable.

4  Right-click the link that connects the variables, and select Properties.

5  In the Link Properties dialog box, click the General tab, and select the Copy only if check box. Developer enables the condition.

6  Click OK.

Modifying the Current Pipeline

During debugging, you can modify the contents of the pipeline and submit those changed values to the next step in the flow service. For example, if you want to see the effect that different values for a variable have on the rest of the service, you can modify the values in the pipeline temporarily and continue debugging. You can also drop values from the pipeline. This functionality is useful for debugging.

When modifying the pipeline, keep the following points in mind:

- You can only modify the pipeline when a subsequent step in the service exists to which to pass the pipeline values. For example, if you use Trace for the entire service, you cannot modify the values of the pipeline after the service ends. However, if you use Step to debug the service, you can modify the pipeline values for the next step in the service.

- You cannot modify the values of Objects, Object Lists, and recursive records. However, you can drop them from the pipeline.

- When you modify values or drop variables from the pipeline, the changes only apply to the current debugging session. The service is not permanently changed.

- You can only modify or drop existing variables. You cannot add new variables to the pipeline.

- You cannot use a larger editor or have a password field when you modify String values in the pipeline.
To modify values in the current pipeline

1. Use the Step, Step Into, or Trace to Here command to load values into the pipeline for the current step.
2. Click the Results tab and select the name of the variable for which you want to change the value.
3. Right-click and select Modify Value.
4. In the Modify Value dialog box, type the new pipeline value for the variable.
5. Click OK. The value is changed in the Results tab.
6. To debug the rest of the service with the new pipeline value, use the Step, Step Into, or Trace to Here command. Keep in mind that the value is only changed for the current debugging session; it is not changed permanently.

To drop values from the current pipeline

1. Use the Step, Step Into, or Trace to Here command to load values into the pipeline for the current step.
2. Click the Results tab and select the name of the variable that you want to drop from the pipeline.
3. Right-click and select Drop. The variable disappears from the Results tab.
4. To debug the rest of the service with the dropped pipeline value, use the Step, Step Into, or Trace to Here command. Keep in mind that the value is only dropped for the current debugging session; it is not dropped permanently.

Saving and Restoring the Pipeline

Because the pipeline contains the data that a service operates against, the ability to save and restore the pipeline when you are debugging a service is something you may frequently want to do. For example, if a service is failing intermittently at run time, you may want to insert steps to save the pipeline at various points in the service so you can capture and examine the data that it was running against after a failure.

Saving the Results

You can save the pipeline to a file, which you can use to restore the pipeline to its current state at a later point in time. This is useful when you want to test another service against the current set of pipeline values or if you want to restore the pipeline to this exact state later in the debugging process. There are two ways to save the contents of the pipeline:
You can manually save the contents of the Results tab when you run or debug a service using Developer.

You can programmatically save the pipeline at run time by invoking `pub.flow:savePipelineToFile` at the point where you want to capture the pipeline.

When you save a pipeline, it is saved in a file in XML format. The file you create can be used to:

- Manually load the pipeline into Developer’s Results tab.
- Dynamically load the pipeline at run time using the `pub.flow:restorePipelineFromFile` service.
- Load a set of input values into the Input dialog box when testing a service with Developer.

You can view a pipeline file with an ordinary text editor. When saving the pipeline, keep the following points in mind:

- Only XML-codable variables are saved. This includes, Strings, String Lists, String Tables, Records, and Record Lists. Variables that are not XML-codable are not saved.
- Empty variables and null variables are saved.

## Saving the Contents of the Results Tab

Use the following procedure to save the contents of the Results tab to a pipeline file.

### To save the contents of the Results tab

1. Display the Results tab and click anywhere in the top area of the tab.
2. Select the Save Pipeline command from the File menu and select one of the following commands:

<table>
<thead>
<tr>
<th>Select this command...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save Pipeline Locally</td>
<td>Save the file to your local file system.</td>
</tr>
</tbody>
</table>

   **Note:** If you intend to use the pipeline file to dynamically restore the pipeline using `pub.flow:restorePipelineFromFile`, use **Save Pipeline to Server** to save the file to the server (see below).

   | Save Pipeline to Server | Save the file to the pipeline directory on SAP BC Server. |

   Select this command if you want to use the file to dynamically restore the pipeline at run time using the `pub.flow:restorePipelineFromFile` service.
Saving and Restoring the Pipeline

Tip! You can also select these commands from the right-click menu on the Results tab.

3 Depending on your action in the previous step, do one of the following:

<table>
<thead>
<tr>
<th>If you selected...</th>
<th>Do this to specify the file name...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save Pipeline Locally</td>
<td>Select the directory in which you want the file saved and assign a name to the file.</td>
</tr>
<tr>
<td>Save Pipeline to Server</td>
<td>Specify the name of the file in which you want the pipeline saved. By default, Developer saves the file to &lt;sapbc&gt;\server\pipeline, which is where the restorePipelineFromFile service expects pipeline files. If you specify a relative path in the file name, the path is understood to be relative to the pipeline directory.</td>
</tr>
</tbody>
</table>

Saving the Pipeline at Run Time

Use the following general steps to save the pipeline programmatically. You can use this technique to capture the pipeline from a flow service or within a Java service.

1 Open the service using Developer.
2 Invoke pub.flow:savePipelineToFile at the point where you want to save a copy of the pipeline.
3 Set the following parameter:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>A string that specifies the name of the file to which you want the file saved. If you do not specify a fully qualified path, the file is saved relative to &lt;sapbc&gt;\server\pipeline.</td>
</tr>
</tbody>
</table>

4 Save the service. (If you are using your own IDE, you will need to recompile the service, reregister it on SAP BC Server, and reload its package.)
5 Execute the service.

For additional information about pub.flow:savePipelineToFile, see the SAP BC Built-In Services Guide.
## Restoring the Pipeline

Pipeline values that you have saved to a file in the following ways can be used to restore the pipeline:

- From the Developer's **Results** tab with the **Save Pipeline** commands.
- With the `pub.flow:savePipelineToFile` service at run time.
- From the **Input** dialog box when you are testing a service with Developer.

Restoring a pipeline is useful when you simply want to inspect the values in a particular pipeline file (perhaps one that contains the pipeline from a failed service). Additionally, it is useful in many testing situations. For example, you can use it to replace the existing pipeline with a different set of values when stepping through a flow service with the debugging tools.

There are two ways to restore the contents of the pipeline:

- You can manually load the saved pipeline into the **Results** tab in Developer.
- You can programmatically load a saved pipeline at run time by invoking `pub.flow:restorePipelineFromFile` at the point where you want to modify the pipeline.

### Loading a Saved Pipeline into the Results Tab

Use the following procedure to load a pipeline file into the **Results** tab.

When you load a pipeline file into the **Results** tab, the contents of the pipeline file completely replaces the current pipeline. If you want to merge the contents of the file with the existing pipeline, use the `pub.flow:restorePipelineFromFile` service instead and set its `merge` parameter to “true.” For procedures, see “Loading a Saved Pipeline at Run Time” on page 279.

---

**To load a pipeline file into the Results tab**

1. Display the **Results** tab. (If you simply want to inspect a saved pipeline, create a new, empty flow service and display its **Results** tab.)

2. Select the **Load** command from the **File** menu and select one of the following:

   - **Select...**
   - **Load Pipeline Locally**
   - **Load Pipeline from Server**

   **To...**
   - Load a pipeline file from your local file system.
   - Load a file that resides in the default pipeline directory on SAP BC Server.

---

**Tip!** You can also select these commands from the right-click menu on the **Results** tab.
3 Depending on your action in the previous step, do one of the following:

If you selected...  
Load Pipeline Locally  
Load Pipeline from Server  

Do this to specify the file name...
Select the file that you want to load.  
Specify the name of the file that you want to load.  
Developer retrieves the file from  
<sapbc>\server\pipeline.

Loading a Saved Pipeline at Run Time
Use the following general steps to load a pipeline file programmatically. You can use this technique from a flow service or from a Java service.

1 Open the service using Developer.
2 Invoke pub.flow:restorePipelineFromFile at the point where you want to load the pipeline file.
3 Set the following parameters:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>A String that specifies the name of the pipeline file. If you do not specify a fully qualified path, the file is assumed to be relative to &lt;sapbc&gt;\server\pipeline. For example, if you set filename to “badPipeline.xml”, restorePipelineFromFile expects to find that file in &lt;sapbc&gt;\server\pipeline\badPipeline.xml.</td>
</tr>
<tr>
<td>merge</td>
<td>A String that specifies whether you want the contents of the file to replace or be merged with the existing pipeline.</td>
</tr>
<tr>
<td>Set merge to...</td>
<td>To...</td>
</tr>
<tr>
<td>false</td>
<td>Replace the existing pipeline with the one from the file.</td>
</tr>
<tr>
<td>true</td>
<td>Merge the contents of the file into the existing pipeline.</td>
</tr>
</tbody>
</table>

4 Save the service. (If you are using your own IDE, you will need to recompile the service, reregister it on SAP BC Server, and reload its package.)
5 Execute the service.

For additional information about pub.flow:restorePipelineFromFile, see the SAP BC Built-In Services Guide.
Other Debugging Techniques

This section describes additional tools and techniques you can use to obtain run-time information that is useful for debugging a service.

Using the Server’s Debug Facility

SAP BC Server maintains a log file in which it records information about activity on the server. This log file resides in:

    <sapbc>/server/logs/server.log

The log contains information about actions that the server executes, such as loading packages and executing services. The following example shows the series of messages that are posted to server.log when the server is started. Note that error messages are posted for services that the server cannot load.

Section of server.log from the start-up process

```
000000  -----------  Tue Jun 10 10:16:00 EDT 2000
000001  SRVINI0001C Integration Server Euphrates Build 336
000002  SRVINI0006C License Manager started
000003  SRVINI0008C State Manager started
000004  SRVINI0010C Service Manager started
000005  SRVINI0012C Cache Manager started
000006  SRVINI0009C WIDL Service Manager started
000007  SRVINI0004C Flow Service Manager started
000008  SRVINI0002C Package Manager started
000033  PCKGES0005C Loading SGxOrders package
000034  SERVIC0002E Failure while loading service SGxOrders.OrderEntry.Bounce:Rejected:...
000035  FLOW S9999W Warning: Flow Branch step with no body in service...
000036  FLOW S9999W Warning: Flow Branch step with no body in service...
000037  FLOW S9999W Warning: Flow Branch step must have switch name or use...
000038  SERVIC0002E Failure while loading service SGxOrders.OrderEntry.Post:PostPO_SGX:...
000039  SERVIC0002E Failure while loading service SGxOrders.OrderEntry.Post:PostPO_Log:...
00002B  SRVINI0005C Port Manager started
00002C  SRVINI0013C Cache Sweeper started
00002D  B2BSRV0002C Initialization complete.
00002E  -----------  Tue Jun 10 10:24:13 EDT 2000
```

The Contents of server.log

The server.log file receives operational and error information from the server’s major subsystems. For example, the package subsystem logs information into server.log when it loads and unloads packages; the flow manager records information in the log when it processes a flow service; the HTTP port records requests that it receives, and so forth.

Be aware that the server does not log exceptions thrown by individual services. However, you can code your service to write information to the server.log, which can be useful for
debugging. For information about writing information to the log file, see “Writing Information to server.log” on page 281.

Server Debug Levels

The amount and type of information that is logged by the server is determined by the debug level under which the server is operating. The debug level is a value from 1 to 10 that you specify when you start the server. The higher the debug level, the more detail the server keeps in its log. For instance, when the server runs under debug level 4, it logs a message for each package that it loads; when it runs under level 5, it logs a message for each individual service within each package that it loads.

The following example shows the startup command you would use to start the server at debug level 7.

```
bin\server.bat -debug 7
bin\server.sh -debug 7
```

(under Windows)

(under UNIX)

If you do not explicitly set the debug switch when you start the server, the default value specified in the `watt.debug.level` parameter is used. SAP ships the server with `watt.debug.level` set at 4.

Once you start the server, the debug level is set. Once the server is running, you can change the debug level using the Server Administrator. If you do not know the debug level under which your SAP BC Server operates, see your SAP BC Server administrator.

**Important!** Debug levels above 6 produce lots of detail and can generate an extremely large log file very quickly. You should not run your server at this level except for brief periods when you are attempting to troubleshoot a particular issue. You may also optionally redirect server.log messages to the console instead of a file using the `-log none` start-up switch. For more information about this switch and debug levels, see “Starting the SAP BC Server” in the *SAP BC Administration Guide*.

Writing Information to server.log

SAP provides built-in services that allow you to write information to server.log at run time. These can be very useful during debugging because you can use them to build signals that indicate whether certain segments of code were executed. You can also use them to record the run-time value of a specific variable.

There are two ways to write information to the server.log at run time. You can:

- Write an arbitrary message to the log using `pub.flow:debugLog`.
- Dump the contents of the entire pipeline to the log using `pub.flow:tracePipeline`.

---

**Other Debugging Techniques**

- [Other Debugging Techniques](#)
CHAPTER 11 Testing and Debugging Services

Writing an Arbitrary Message to the Log

To write an arbitrary message to the server log, you invoke the built-in service pub.flow:debugLog. You can invoke pub.flow:debugLog from a flow service or a coded service (such as a Java service). When this service executes, it puts a text string that you specify into the server.log. You might use it to post progress messages at certain points in a service (to indicate whether certain segments of code were executed) or to record the value of a particular variable to the log file so you can examine it after the service executes.

The following example shows two progress messages (highlighted) that were posted to the server log using pub.flow:debugLog.

Example of messages posted to server.log with pub.flow:debugLog

To use pub.flow:debugLog, take the following general steps:

1. Invoke pub.flow:debugLog at the point where you want the service to write a message to the server log.
2. Set the following parameters:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>A String that specifies the message that you want written to server.log. This can be a literal string that you assign to message, however, for debugging purposes, it is often useful to map this parameter to a pipeline variable whose run-time value you want to capture.</td>
</tr>
</tbody>
</table>
Save the service. (If you are using your own IDE, you will need to recompile the service, reregister it on SAP BC Server, and reload its package.)

4 Execute the service.

For additional information about pub.flow:debugLog, see the SAP BC Built-In Services Guide.

Dumping the Pipeline to the Log

Sometimes when you are debugging a service, it is useful to obtain a snapshot of the entire pipeline at a certain point in the flow. You can do this by invoking pub.flow:tracePipeline, which puts a copy of the current pipeline in server.log. You may invoke pub.flow:tracePipeline from a flow service or a coded service (such as a Java service).

The following example shows a pipeline (highlighted) that was written to the server log with pub.flow:tracePipeline.

### Key Description

- **function**  
  **Optional.** A String that identifies your service. The String you specify will appear in the second column of the message that debugLog writes to server.log. The purpose of this label is to identify which component posted the message to the log.

  If you do not assign a value to function, debugLog uses the string “------” as the label. However, keep in mind that assigning a value to function will make it easier for you to locate your service’s message when you examine the log file. Although you can assign a text string of any length to function, only the first 6 characters appear in the log.

- **level**  
  **Optional.** A String containing a value from 1 to 10, specifying the debug levels under which this message is to be posted to the log. If the server is running at a debug level lower than the value set in level, the message is not put into the log file.

  If you do not specify level, 1 is assumed, which means that the message is posted to the log file regardless of which debug level the server is running at. For more information about debug level, see “Server Debug Levels” on page 281.

3 Save the service. (If you are using your own IDE, you will need to recompile the service, reregister it on SAP BC Server, and reload its package.)

4 Execute the service.
Example of pipeline written to server.log with pub.flow:tracePipeline

```plaintext
00065F HTTPREQ0001V POST /invoke/SGXOrders.OrderEntry.Collect/PoMDW_SGxOrderRec
000660 FLWOPR00005V Invoke : index=1 depth=1
00066B FLWOPR00005V Invoke : index=1 depth=1
00066C FLWOPR00005V Invoke : index=1 depth=1
00066D FLWOPR00005V Invite : index=1 depth=1
00066E FLWOPR00005V Invite : index=1 depth=1

0006F0 PIPE--9999D 0 message = 05082000 Trace after SuperMap
0006F1 PIPE--9999D 0 Marker1 = marker1
0006F2 PIPE--9999D 0 function = SMAPXXX
0006F3 PIPE--9999D 0 level = 4
0006F4 PIPE--9999D 0 node = DOCUMENT System ID: null Public ID: null
0006F5 PIPE--9999D 0 filename = d:\test\file.xml
0006F6 PIPE--9999D 0 IntlDate = 05082000
0006F7 PIPE--9999D 0 PO_In =>
0006F8 PIPE--9999D 1 @version = 1.0
0006F9 PIPE--9999D 1 PO =>
0006FA PIPE--9999D 2 poNum = 99401088
0006FB PIPE--9999D 2 orderDate = 08/05/2000
0006FC PIPE--9999D 2 buyerInfo =>
0006FD PIPE--9999D 3 companyName = Midwest Extreme Sports, Inc
0006FE PIPE--9999D 3 TPID = MDW_A1099847
0006FF PIPE--9999D 3 accountNum = GSG-970414-0A
000700 PIPE--9999D 3 phoneNum = (612) 495-3300
000701 PIPE--9999D 3 faxNum = (612) 495-3337
000702 PIPE--9999D 3 addr =>
000703 PIPE--9999D 4 Addr1 = #111
000704 PIPE--9999D 4 Addr2 = 15788 Cedar Ave
000705 PIPE--9999D 4 city = Apple Valley
000706 PIPE--9999D 4 state = MN
000707 PIPE--9999D 4 postalCode = 55124
000708 PIPE--9999D 4 Email = buyers@GSG.com
000709 PIPE--9999D 3 Buyer = Caroline Wielman
00070A PIPE--9999D 2 supplier =>
00070B PIPE--9999D 3 Name = Global Sporting Goods
00070C PIPE--9999D 3 vendorNum = V98745645
00070D PIPE--9999D 3 TPID = GSG_89333789G6F
00070E PIPE--9999D 2 order =>
00070F PIPE--9999D 3 lineItem[0] =>
000710 PIPE--9999D 4 ItemNum = 965007
000711 PIPE--9999D 4 desc = MaxGear D Quick Lock Carabiner
000712 PIPE--9999D 4 qty = 500
000713 PIPE--9999D 4 uPrice = 2.10
000714 PIPE--9999D 4 taxCode = T
000715 PIPE--9999D 3 lineItem[1] =>
000716 PIPE--9999D 4 stockNum = 965003
000717 PIPE--9999D 4 desc = MaxGear D LtWt D Carabiner
000718 PIPE--9999D 4 qty = 300
000719 PIPE--9999D 4 uPrice = 8.50
00071A PIPE--9999D 3 lineItem[2] =>
00071B PIPE--9999D 4 stockNum = 896301
00071C PIPE--9999D 4 desc = Makas 10.5x50 Standard Rope
00071D PIPE--9999D 4 qty = 50
00071E PIPE--9999D 4 uPrice = 175.00
00071F PIPE--9999D 2 fob =>
000720 PIPE--9999D 3 type = ground
000721 PIPE--9999D 3 Addr1 = #111
000722 PIPE--9999D 3 Addr2 = 15788 Cedar Ave
000723 PIPE--9999D 3 city = Apple Valley
000724 PIPE--9999D 3 state = MN
000725 PIPE--9999D 3 postalCode = 55124
000726 PIPE--9999D --- END tracePipeline ---

000727 FLWOPR00005V Invite : index=1 depth=1
000728 FLWOPR00005V Invite : index=1 depth=1
```
To use pub.flow:tracePipeline, take the following general steps:

1. Invoke pub.flow:tracePipeline at the point where you want the service to dump a copy of the pipeline to the server log.

2. Set the following parameters:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>Optional. A String containing a value from 1 to 10, specifying the debug levels under which the pipeline will be posted to the log. If the server is running at a debug level lower than the value set in level, the pipeline is not written to the log file. If you do not specify level, 4 is assumed, which means that the pipeline will not be dumped to the log file unless the server is running at debug level 4 or higher. For more information about debug level, see “Server Debug Levels” on page 281.</td>
</tr>
</tbody>
</table>

3. Save the service. (If you are using your own IDE, you will need to recompile the service, reregister it on SAP BC Server, and reload its package.)

4. Execute the service.

For additional information about pub.flow:tracePipeline, see the SAP BC Built-In Services Guide.
Building Coded Services

- Basic Concepts ................................................................. 288
- Building Services Using Java ........................................ 290
- How Java Services Are Organized on the Server ............. 290
- Building Java Services with Developer .......................... 291
- Building Java Services with Your Own IDE ................. 298
- Debugging a Java Service ................................................. 304
- Building Services Using C/C++ ..................................... 306
- Building Services Using COM ....................................... 310
- Invoking Methods from Existing COM and DCOM Objects . 311
- Writing a Visual Basic Service ...................................... 312
- Invoking a Visual Basic Service .................................... 313
CHAPTER 12  Building Coded Services

Basic Concepts

In addition to using the built-in services that SAP provides, you can create customized services in a variety of program languages. This allows you to create a library of custom code that can be accessed and executed from a flow service or from a client application.

This chapter describes how to create your own services using Java, C/C++, and Visual Basic.

**Important!** Java is the native language for services. When you create services in other languages, you must wrap them to appear as a Java class to SAP BC Server.

The IData Object

The IData object is the universal container that services use to receive input from and deliver output to other programs. It contains an ordered collection of key/value pairs on which a service operates. An IData object can contain any number of key/values pairs (elements). The keys in an IData object must be Strings. The values can be any Java objects (including IData objects).

**Services Take IData Objects as Input and Return IData as Output**

Services take one, and only one, input variable—an IData object. For example:

```java
public final static void myservice (IData pipeline)
    throws ServiceException
{
    return;
}
```

When a service is invoked, SAP BC Server passes the IData object to it. The service extracts the actual input values it needs from the elements within the IData object. For example:

```java
public final static void myservice (IData pipeline)
    throws ServiceException
{
    IDataCursor myCursor = pipeline.getCursor();
    myCursor.first( "inputValue1" );
    String myVariable = (String) myCursor.getValue();
    myCursor.destroy();
    return;
}
```

If you use the utility class IDataUtil, the coding you don’t have to deal with the cursor directly. Sometimes this might be useful and the coding above would look as follows, with the slightly different semantics that it makes sure that a key in the pipeline remains unique, when adding new values:

```java
public final static void myservice (IData pipeline)
    throws ServiceException
{
    IDataCursor myCursor = pipeline.getCursor();
    ...
A service returns output by inserting it into an IData object. Any information that is produced by the service and must be returned to the calling program, needs to be written to the IData object. For example:

```java
public final static void myservice (IData pipeline)
    throws ServiceException
{
    IDataCursor myCursor = pipeline.getCursor();
    myCursor.first("inputValue1");
    String myVariable = (String) myCursor.getValue();
    myCursor.last();
    myCursor.insertAfter("outputValue1", myOutputVariable);
    return;
}
```

**Getting and Setting Elements in an IData object**

Getting data from and putting data into IData elements takes two steps. First, you must position the cursor at the IData element. Next, you get or set the data in that element.

There are several different classes you can use to position a cursor in an IData object.

<table>
<thead>
<tr>
<th>This Cursor Class</th>
<th>Contains methods for...</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDataCursor</td>
<td>Performing basic cursor operations such as placing the cursor at the first, last, or next element in the object and positioning the cursor at a specified key. The latter functionality was originally only provided by an IDataHashCursor.</td>
</tr>
<tr>
<td>IDataIndexCursor</td>
<td>Positioning the cursor by absolute position.</td>
</tr>
</tbody>
</table>

After you position the cursor on the element with which you want to work, you can use the getValue or setValue methods to read or write the value of that element, respectively. These classes also provide methods for inserting new elements, getting key names, and deleting elements.

For more information about using the cursor classes, see the data package in the SAP BC Java API Reference at <sapbc>/developer/doc/api/index.html.

**Creating IData objects**

You use the IDataFactory class to create IData objects. For example:

```java
static IData myIDataObject;
static
{
    myObject = IDataFactory.create();
    IDataCursor myCursor = myObject.getCursor();
}
```
myCursor.last();
myCursor.insertAfter("VA", new Double("0.045"));
myCursor.insertAfter("MD", new Double("0.05"));
myCursor.insertAfter("DE", new Double("0.0"));
}

For more information about using the IDataFactory class, see the data package in the SAP BC Java API Reference at: <sapbc>/developer/doc/api/index.html.

The Values Object

Earlier versions of SAP BC Developer (prior to SAP BC Developer 3.5) used Values objects as the basic data structure. Now, the Values object is an implementation of the IData object. (Values objects contain methods in addition to those in the IData interface, which provide backward compatibility.)

When using the Values API, be aware that it does not preserve element order or duplicate keys. The use of Values objects for development of new services is discouraged. IData objects should be used instead. Apart from preserving order and supporting duplicate keys, the native IData interface provides better performance than the Values API.

Building Services Using Java

Since Java is the native language of services, it is the easiest language in which to build a service.

SAP BC Developer provides an Integrated Development Environment (IDE) that you can use to create, compile, and publish Java services. The IDE automatically generates an appropriately-structured source file that you simply “fill-in” using the built-in editor. When you save the source file, the IDE automatically compiles it and registers it on the server.

Although Developer's IDE is useful for creating small, simple services, you may want to use your own Java IDE to build large services. When you use your own IDE, you must create all of the code yourself, and manually copy and register the class file on the server. SAP BC Server provides utilities to publish services you write in your own IDE. For more information about creating Java services without using Developer, see “Building Java Services with Your Own IDE” on page 298.

How Java Services Are Organized on the Server

A Java service is a public static method in a Java class file on SAP BC Server. They follow a simple naming scheme:

- The service name represents the Java method name
- The folder name represents the fully qualified Java class name. As of Business Connector 4.8 the class name itself can be prefixed with ‘JSBC_’. This avoids name
clashes of classes with Java packages if you have another nested folder that contains Java Services as well.

Since Java class names cannot contain the “.” character, services that reside in nested folders are implemented in a class that is scoped within a Java package. For example, a service named recording.accounts:createAccount is made up of a Java method called createAccount in a Java class called accounts within the recording package. In case of a prefixed class name the class name would be .JSBCaccounts.

All Java services that reside in the same folder are methods of the same class.

When you build a Java service with Developer, it automatically combines your service into the class file associated with the folder in which you created it. However, if you build a Java service in your own IDE, you will need to add the class file to the server yourself. And, if you want that service to be recognized by Developer, you must insert special comment code in your source code.

Building Java Services with Developer

The following describes the basic stages involved in creating a Java service with Developer.

**Stage 1** Specify the inputs and outputs of the service. This stage is optional, but recommended. During this stage, you define the service’s inputs and outputs (if you know these) in Developer’s IDE. For information about this stage, see “Generating Java Code from Service Input and Output Parameters” on page 296.

**Stage 2** Create the Java service using Developer. During this stage, you write your program in Developer’s IDE. For information about this stage, see “Creating a Java Service with Developer’s IDE” on page 295.

**Stage 3** Specify the service’s run-time parameters. During this stage, you assign parameters that configure the run-time environment for this service. For information about this stage, see “Setting Run-Time Options for a Java Service” on page 297.

Before you create a Java service, you must:

- **Make sure the package in which you want to create the service already exists.** If the package does not already exist, use Developer or the Server Administrator to create it. See online help for step-by-step procedures.

- **Make sure the folder in which you want to create the service already exists.** If the folder does not already exist, use Developer to create it. See Developer’s online help for step-by-step procedures.
Make sure that all Java, C, and WebTap services are unlocked or locked by you in the folder in which you want to create the new service. For details, see the SAP BC Cooperative Development Guide.

**Important!** All Java services that belong to the same folder reside in the same Java class file. This class has the same name as the folder.

### Using the Developer IDE

In the Developer IDE, you use two tabs to create your source code: the **Source** tab and the **Shared** tab.

#### The Source Tab

You use the **Source** tab to build the body of your program. It is like a template you “fill-in” with custom Java code. Standard blocks of required code appear in the shaded areas at the top and bottom of the tab. You cannot alter the code in these areas.

The **Source** tab is like a template for building a service

The required code at the top of the **Source** tab defines a static and final method with a single input parameter—an IData object. The block of required code at the bottom returns the pipeline to the caller.
When you build a Java service, you type (or paste) your code in the text box on the Source tab. The following example shows a service that gets two values from the pipeline and uses them to compute a sales tax. It puts the computed tax into the pipeline.

You use the Source tab to write the body of your service:

```java
SalesTax

    static final public void SalesTax(Data pipeline | throw ServiceException)
    {
        // pipeline
        DataHashCursor pipelineCurrent = pipeline.getHashCursor();
        pipelineCurrent.first();
        String state = (String) pipelineCurrent.getValue();
        String amount = (String) pipelineCurrent.getValue();
        double subtotal = Double.valueOf(amount).doubleValue();
        TaxState ratesHashCursor = rates.getHashCursor();
        double rate = 0;
        if (ratesHashCursor.first(state))
            rate = (double) ratesHashCursor.getValue().doubleValue();
        ratesHashCursor.destroy();

        // pipeline
        DataHashCursor pipelineCurrent_1 = pipeline.getHashCursor();
        pipelineCurrent_1.first();
        pipelineCurrent_1.normalize("Total");
        double totalingSubtotal = rate;
        pipelineCurrent_1.destroy();

        subtotal = rate; // subtotal is the subtotal of the tax.
        pipelineCurrent.setValue(subtotal);

        pipelineCurrent.destroy();
    }
}
```

The Shared Tab

You use the Shared tab to specify common (i.e., shared) attributes of this class. This includes the superclass and interface declarations, required imports, and member variables that are shared but not exposed as services. The code on this tab is shared by all services in this folder.
You use the Shared tab to specify the common attributes of the class

<table>
<thead>
<tr>
<th>SalesTax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extends</td>
</tr>
<tr>
<td>Implements</td>
</tr>
<tr>
<td>Imports</td>
</tr>
<tr>
<td>Source</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

The **Extends** field allows you to specify a super class for the implementation. You are not required to specify a super class.

**Note:** It is useful, but not necessary, to extend the class `com.wm.app.b2b.server.Service`. This class includes static methods for various common tasks, like retrieving the current session ID and formatting error messages.

The **Implements** field specifies the names of Java interfaces that you want to implement in the extended class.

The **Import** field specifies the names of additional Java packages whose classes are available to the current class. When you create a Java service with Developer, several Java packages are automatically added to the import list.

The **Source** field allows you to define global variables and methods that are shared by all services in the current folder. This is useful for building shared data structures and supporting functions that are not intended to be exposed as services. For example, you might use the **Source** field to define an account table and the methods to used to access it for a set of services that create, get, and delete account information.

**Note:** Because services are implemented as static methods, most shared code is usually static as well.
Creating a Java Service with Developer’s IDE

The following procedure describes how to create the source code for a Java service using Developer’s IDE.

To create a Java service with Developer

1. On the File menu, click New.
2. Select Java Service and click Next.
3. In the New Java Service dialog box, next to Folder, select the folder into which you want to save this service.
4. In the Name field, type a name for the service.
5. Click Finish.
6. If you know the set of inputs and outputs your program uses, specify these using the Input/Output tab.

Note: You can use Developer to automatically generate Java code that gets and puts those input and output values in the pipeline. For more information about automatically generating Java code, see “Generating Java Code from Service Input and Output Parameters” on page 296.

7. Type the code for your service on the Source tab. For information about Java classes provided by SAP, see the SAP BC Java API Reference in <sapbc>/developer/doc/api/index.html.
8. If you want to make additional methods and/or structures available to the service, complete the following fields on the Shared tab.

Use this field... To specify...

Extends The name of the superclass (if any) of which this class is an extension. If you specify a superclass, type its Java class name (fully qualified if necessary).

Implements The names of interfaces within the superclass that this class implements. Take the following steps to specify each interface that you want to implement:

1. Click to add a new row to the list.
2. Type the name of a valid Java class name (fully qualified if necessary). You do not need to type the “implements” keyword.
When you finish specifying your code on the Source and Shared tabs, click on the toolbar to save and compile the service.

Generating Java Code from Service Input and Output Parameters

If, before you start writing your service, you know the set of inputs and outputs that it will use, you can declare the service’s input/output parameters first and generate Java code from it. This code gets the specified input values from the pipeline and assigns them to variables in your program. It also puts the output values into the pipeline.

For example, if the Input/Output tab for the service defines the following variables as input and output:

<table>
<thead>
<tr>
<th>Input Variable Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>String</td>
</tr>
<tr>
<td>Amount</td>
<td>String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Variable Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tax</td>
<td>String</td>
</tr>
</tbody>
</table>

Developer will generate the following Java code for your service:

```java
// pipeline
IDataCursor pipelineCursor = pipeline.getCursor();
StringState = (String) pipelineCursor.getValue();
pipelineCursor.first("State");
StringAmount = (String) pipelineCursor.getValue();
pipelineCursor.destroy();

// pipeline
IDataCursor pipelineCursor_1 = pipeline.getCursor();
pipelineCursor_1.last();
pipelineCursor_1.insertAfter("Tax", "Tax");
pipelineCursor_1.destroy();
```
When Developer generates code from the service input/output parameters, it puts the code on the Clipboard. From there, you can paste it into your program (on the Source tab or in your own IDE) and modify it as necessary.

**Note:** SAP BC Server returns everything that your service puts into the pipeline, regardless of what is declared as the its input/output parameters. Declaring a service’s input and output parameters does not filter what variables the service actually receives or returns at runtime. It simply provides a formal description of what the service requires as input and produces as output.

The following procedure describes how to generate code from a the input/output parameters defined for a service.

**To generate Java code from the service input/output parameters**

1. Open the Java service for which you want to generate code (if you are creating the Java service in your own IDE, use Developer to create a new, empty Java service that you will use only for the purpose of declaring a set of input/output parameters).
2. Select the Input/Output tab and define the inputs and outputs for this service if they are not already specified. For more information about defining inputs and outputs for a service, see “To declare input and output parameters for a service” on page 86.
3. If you want to generate code for a subset of the variables on the Input/Output tab, select those variables by pressing CTRL and clicking the variable names with your mouse.
5. Select For Implementing This Service and click Next.
6. Specify the following options.

<table>
<thead>
<tr>
<th>Under this...</th>
<th>Specify...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Which Records?</td>
<td>Whether you want to generate code for the input variables, the output variables, or both.</td>
</tr>
<tr>
<td>Which Fields?</td>
<td>Whether you want to generate code for all variables in the Input/Output tab or just the selected variables.</td>
</tr>
</tbody>
</table>

7. Click Finish. Developer generates code and places it on the Clipboard.
8. Paste the contents of the Clipboard into your source code.

**Setting Run-Time Options for a Java Service**

When you create a Java service with Developer, you can set options to specify the runtime behavior of the service. These options determine:
Whether the service runs in stateless mode.
Whether the results of the service are maintained in cache.
Whether an output template is applied to the service when it is invoked by a browser.
You specify these options on the Settings tab. For information about using these options, see “Specifying Run-Time Parameters” on page 87 and “Assigning an Output Template to a Service” on page 91.

Building Java Services with Your Own IDE

There may be times when you want to use your own IDE instead of Developer to build a Java service.

The Namespace Directory

To successfully publish (install) a Java service that you have created with your own IDE, you need to understand exactly how Java services are stored on SAP BC Server.

Each package on SAP BC Server has a namespace directory, called “ns.” For example, a package called “purch” would have the following directory structure:

```
<sapbc>\server\packages\purch\ns
```

The ns directory contains information about the services in that package. An “entry” in the namespace directory corresponds to a service or a folder. The contents of each entry depend on what kind of entry it is.

- **Service entries** contain information about properties of the service (for instance, statelessness), the input and output parameters of the service (if they have been defined), as well as Java or XML source if it's available for that service.
- **Folder entries** contain information about the folder—this is usually limited to Java source for the services in that folder, if it's available.

For Java services, information about the service is stored in the namespace directory, however the compiled code for that service (i.e., the class file) is stored in the code subdirectory. The following shows the directory path to the class files in the purch package.

```
<sapbc>\server\packages\purch\code\classes\recording\accounts.class
```

When you use Developer to build a Java service, it automatically updates and maintains the folder and service information in the namespace. However, if you build a Java service in your own IDE, you need to use a utility called jcode to compile your service and generate the necessary files in the namespace.
The Source Code Directory

Each package on the server has a source subdirectory that holds the Java source code for that package, if it is available.

When you finish coding your Java service, save its source file in this directory (subject to the normal Java constraints based on package declarations). You must name the files and intermediate directories according to the name of the service you are installing. For instance, the source file for the recording accounts services shown in Appendix F would have the following path:

```<sapbc>\server\packages\purch\code\source\recording\accounts.java```

Writing the Source Code for a Service

Keep in mind that your service must be written as a method that takes an IData object as input.

A Java service is a method that is public and static. It takes a single instance of com.wm.data.IData as input and returns output in the pipeline. The following code shows the basic framework for a Java service:

```java
public final static void myservice(IData pipeline)
    throws ServiceException
{
    return;
}
```

**Note:** Services can throw ServiceException. Do not call Service.throwError.

Additionally,

- Your Java class must import the following Java packages.
  ```java
  com.wm.data.*;
  com.wm.app.b2b.server.ServiceException;
  com.wm.app.b2b.server.Service;
  ```
- Your Java class must be public.
- We also recommend, for performance reasons, that you make your class final.
Using the SAP BC API

SAP provides classes that you can use with Java services that you build. See the SAP BC Java API Reference in:

<sapbc>/developer/doc/api/Java/index.html

for a description of these classes.
The Basic Stages

The following describes the basic stages involved in creating a Java service with your own IDE.

**Stage 1**
Create an empty Java service using the Developer (optional). During this stage, you can use the Developer to create an empty Java template that you can use a guideline for coding your own service (see “Creating a Java Service with Developer’s IDE” on page 295).

**Stage 2**
Specify the input and output parameters (signature) of the service. During this stage, you define the service’s inputs and outputs (if you know these). You can use the Developer to generate the input and output code that you can paste into your program (see “Generating Java Code from Service Input and Output Parameters” on page 296).

**Stage 3**
Create the Java service using your IDE. During this stage, you write and compile your program in your IDE. For more information about this stage, see “Writing the Source Code for a Service” on page 299.

**Stage 4**
Saving and compiling your code using jcode. Optional. During this stage, you can make your source code available for editing by the Developer by using the jcode utility. For information, see “Commenting Code for the SAP BC Server” on page 301.

**Stage 5**
Publish the service to the SAP BC Server. During this stage, you register your service on the SAP BC Server using the jcode utility. For information, see “Using the jcode Utility” on page 302.

Commenting Code for the SAP BC Server

To install your finished service on the SAP BC Server, you use the jcode utility. To use this utility successfully, you must annotate your source code with jcode tags (specially-formatted Java comments) to “mark” the following segments of code:

- Imports
- Shared code within the class
- Service definitions and service inputs and outputs

The following code fragment shows the tags used to mark the beginning and end of the import section.

```java
// --- <<B2B-START-IMPORTS>> ---
import com.wm.data."
import java.util."
// --- <<B2B-END-IMPORTS>> ---
```
You use similar tags to mark the beginning and end of other components in your source code. For a complete example, see “jcode Examples” in Appendix F. For additional information about the jcode utility, see the next section “Using the jcode Utility”.

Using the jcode Utility

When you finish creating and annotating your source code, you use the jcode utility to compile it and store its service information in the ns directory.

Jcode operates in three basic modes:

- **Make Mode** (for compiling Java source code).
- **Fragment Mode** (for pulling apart source code and storing fragments and signatures in the namespace).
- **Composite Mode** (for rebuilding the source files from fragments in the namespace).

You must use the make and fragment modes to run your services on SAP BC Server and edit the source code from Developer.

**Important!** Before you can compile a service using jcode, you need to edit it and replace the string `%INSTALLDIR` with the directory in which SAP BC Server is installed. On Windows additionally remove the double quotes ("’) in the line SET JAVA_ROOT="$B2B_ROOT\jvm".

Make Mode

You use this mode to examine source files for one or more folders in a package and compile those that have been modified since they were last compiled. The jcode utility will report which files were compiled, as well as any errors that were encountered during the process.

To make all the code in a package, type the following on the command line:

```
jcode makeall Package
```

To compile source files, the jcode utility invokes the JDK Java compiler, javac using the following command:

```
javac --classpath pathName -d classDir fileList
```

Where `pathName` is the classpath to use for the compile, `classDir` is the destination directory for the compiled classes, and `fileList` is a list of the names of source files to compile.

If you do not have the JDK installed, or you want to use another compiler, you can set SAP BC Server’s `watt.server.compile` property to a new command line (using the arguments described above). For instance, to use IBM’s `jikes`, you would set this property to:

```
jikes +E -nowarn --classpath pathName -d classDir fileList
```
Fragment Mode

You use this mode to update the Java code fragments and service signatures (input and output parameters) in the namespace based on the jcode tags in the source code file. The original source file is not modified, but namespace information is updated and the source code for the service becomes available through Developer.

To fragment all the code in a package, type the following on the command line:

```
jcode fragall Package
```

To fragment only the code for a single folder (i.e. a single Java source file), type the following on the command line:

```
jcode frag Package Folder
```

Composite Mode

Composite mode is the opposite of fragment mode. You use this mode to build a source file based on the code fragments currently defined in the namespace.

**Important!** The existing source file, if there is one, is overwritten by the source file produced by jcode. User locks in Developer will not prevent this, since the jcode utility operates independently of locking functionality.

To construct a source file based on the current information in the namespace, type the following on the command line:

```
jcode comp Package Folder
```

**Important!** If your Java source code contains any non-ASCII characters, set the property `watt.server.java.source=Unicode | UnicodeBig | UnicodeLittle`. The default value is `file.encoding`. When Unicode is set, the compile command line specified in the property `watt.server.compile.unicode` is used. The default value of this property is

```
"javac -encoding Unicode -classpath {0} -d {1} {2}"
```

To construct all source files for a package based on the current information in the namespace, type the following on the command line:

```
jcode compall Package
```

This command is especially used for the migration of packages from a release prior to Business Connector 4.8, which didn’t have the concept of the ’JSBC_’ prefix in order to avoid name clashes for the compiler.
Other jcode Commands

Because the two-step process of making and fragmenting source code is so common, there are several shortcuts built in to jcode.

The “update” mode makes and fragments only source files which have changed. To update (make and frag) all the folders within a package which have changed, type the following at the command line:

```
jcode update Package
```

To update (make and frag) all the code in all packages on SAP BC Server, type the following at the command line:

```
jcode upall
```

To force a make and frag on all packages on SAP BC Server, type:

```
jcode hailmary
```

Debugging a Java Service

When you build a Java service, it is often useful to execute it through a debugger to locate and correct any errors in your code. As described in this section, this task involves two basic steps:

- First, you must generate the class file in a way that it can be debugged and register it on SAP BC Server.
- Then, you must attach a debugger to SAP BC Server and execute the service that you want to debug.

**Note:** As of BC 4.8, it is possible to attach a debugger without restarting the server, if the server is running with the SAP JVM. This is a very helpful feature for development purposes, even for tracking problems on productive systems, because you do not need to restart the server.

Preparing Debugging

If you develop your own services, make sure to compile them with the debug option –g so that all relevant information is included in the class files.

For development servers perform the following steps:

1. **Create the java service in BC Developer (including the signature).**
   
   This will create the necessary BC internal files correctly.

2. **Start your IDE**
   
   The following example refers to Eclipse, but can also be adapted to other IDEs.
Debugging a Java Service

Create a project in your IDE which uses the `<sapbc>/server/packages/<yourPackage>/code/source directory`. Recompile the source directory, otherwise you might be getting later errors like "hot code replace failed" in Eclipse.

3 **Put the class file produced by compiler in the appropriate "classes" directory on SAP BC Server.**
   (If you compile your Java service from Developer, you can skip this step, because this happens automatically.) To do this, locate the package in which you want the service to reside, and then copy generated the class file to the `code/classes` directory in that package or set the build output directory to:
   `<sapbc>/server/packages/<yourPackage>/code/classes`

   For example, if you have a class file called `orders.class`, you would copy it here:
   `<sapbc>/server/packages/<yourPackage>/code/classes/order.class`

   Be aware that a class name represents a folder in SAP BC Server. If you want to nest the class file within a sub-folder, you must put it in the appropriate subdirectory within `code/classes`. For example, if you wanted `orders.class` to represent a sub-folder within an folder called `SGX`, you would copy it here:
   `<sapbc>/server/packages/<yourPackage>/code/classes/SGX/order.class`

   For productive servers (in order to find a problem) do the following:
   Create a project in Eclipse that contains the sources of the services you like to debug. Thus it is possible to set a breakpoint in the services that you want to investigate.

4 **If the service you want to debug is not already registered on the server, register it using the Server Administrator.** (If the service was initially created with Developer, it will already be registered on the server.) For information about registering a service, see the SAP BC Administration Guide.

**Running the Debugger**

The following general steps describe how to run a debugger against a Java service on a SAP BC Server 4.8 running on SAP JVM:

1 Open **Server -> JVM -> Debugger Control** in the Server UI.

2 Press **Attach Debugger** (BC then waits for a debugger to attach)

3 Go to Eclipse, open your debug configurations, create a Remote Java Application configuration, set the host and the port, and Press **Debug**.

4 Refresh the Server UI. You should now see that the Debug State changed to **STATE_DEBUGGER_ATTACHED**. Eclipse is now connected, i.e. you can set breakpoints in the source and use the debugging features of Eclipse.

5 Run your service, e.g. in the SAP BC Developer.
If you have set a breakpoint, Eclipse will display the corresponding code and you can analyze the execution state. If another package than the investigated one is passed, you can easily attach its sources to the debug configuration on the fly.

When finished, detach the debugger in Eclipse or in the Server UI.

Building Services Using C/C++

You can use Developer to build a set of starter files you can use to create a C/C++ service. These files include:

- A Java service that calls your C program.
- A C/C++ source-code “template” that you use to create your C program.
- A make file you use to compile the finished program and place it on the server.

Before you create a C/C++ service, you must:

- **Make sure you have a C compiler** installed on SAP BC Server that you will use to develop and test the service.
- **Make sure the package in which you want to create the service already exists.** If the package does not already exist, create it using SAP BC Developer. For more information about creating a package, see “Creating a Package” on page 57. (If you do not have Developer or Administrator privileges, ask your server administrator to do this.)
- **Make sure the directory for this package contains a “code/libs” directory.** When you compile your C/C++ service, the make file places the compiled service (a DLL) in this directory. If the package does not already have a code/libs directory, create one before you begin building the service.
- **Make sure the folder in which you want to create the service already exists.** If the folder does not exist, use Developer to create it. See online help for step-by-step procedures.
- **Declare the input and output parameters for your service in a specification.** When Developer generates the starter code for your service, it creates code that extracts the specified input values from the pipeline and assigns them to variables in your program. It also inserts your service's output variables into the pipeline. To do this, Developer must know the input and output requirements of your service. You supply this information in a specification. For information about creating a specification, see online Help.
Generating Files for a C/C++ Service

If you have satisfied the prerequisites identified above, you can use the following procedure to generate the files that you need to build a C/C++ service.

To generate C/C++ project files

1. On the File menu, click New.
2. Select C Service and click Next.
3. In the New C Service dialog box, in the list next to Folder, select the folder into which you want to save this service.
4. In the Name field, type a name for the service and click Next.
5. Select the platform that describes the machine on which your SAP BC Server is running (Developer needs to know this in order to build the right make file).
6. Select the specification for this service.
7. Click Finish.

The Java Code for a C Service

When you build a C/C++ service, Developer builds a Java service that calls a DLL, which you create by writing a C program. The Java service is the means by which your C program is exposed to clients (IS clients invoke this Java service, not the C program directly). The Java service also supplies the input/output parameters for the program, which makes it possible to include it in a flow service and map its inputs and output with the Pipeline Editor.

Developer generates all the Java code needed to successfully call your C program. You may add your own custom code to the Source or the Shared tab if you want to execute any special procedures before or after the C program is called, but other than that, this service contains everything you need.
CHAPTER 12 Building Coded Services

The Source tab contains code that calls the Java wrapper for the C program

```java
postPOSc:
static final public Values postPOSc( Values in )
{
    Values out = in;
    out = postPOSc(Service.get_session(), in);
}
```

The Shared tab contains code that loads the library containing the C program

```java
postPOSc:
static {
    try {
        System.loadLibrary("purchasing");
        Debug2.log(Debug2.LOG_INFO, Debug2.PAC_PACKAGE,
            "Loading native library: purchasing");
    } catch (UnsatisfiedLinkError e) {
        Debug2.log(Debug2.LOG_INFO, Debug2.PAC_PACKAGE,
            "Loading native library: Purchasing");
    }
```
The Input/Output tab declares the input/output parameters for the service.

Building the C/C++ Source Code

Developer also generates a source code file and a make file for you. It places these files in the following directory:

```
<sapbc>\server\packages\packageName\code\source
```

The names of the files will match the service name you specified in Developer. After you locate the files, do the following:

- **Copy the source code file** to a new file (in the same directory) with the following file name:
  
  `serviceNameImpl.c`

**Example**

If your service name is `postPO`, you would...

```
Copy... To...
postPO.c    postPOImpl.c
```

You create the program in the `serviceNameImpl.c` file, not the original file. This is the file in which the make file expects to find your source code. (This step is taken to maintain a copy of the original source file to which you can refer, or revert back to, during your development.)
CHAPTER 12 Building Coded Services

- **Edit the make file** to customize it for your development environment. Make sure to set the following path settings:

<table>
<thead>
<tr>
<th>Set...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDKDIR</td>
<td>To the directory that contains the Java Development Kit.</td>
</tr>
<tr>
<td>SEVRDIR</td>
<td>The directory in which SAP BC Server is installed (e.g., c:&lt;sapbc&gt;\server).</td>
</tr>
</tbody>
</table>

**Writing the Service**

Edit the `serviceNameImpl.c` file as necessary to build your service. This file will contain instructive comments that will guide your development. You can also refer to SAP BC C API for information about how to use the SAP BC C/C++ API to make the data in your service available to other services. This file is located in `<sapbc>\developer\doc\api\c\index.html`.

**Compiling the Service**

After you finish coding your service, run your make file to compile it. Following is a typical make command:

```
make -f SalesTax.mak
```

The make file compiles your program and puts the finished DLL in the code\libs directory in the package in which the service resides (if this directory does not exist when you run the make file, your program will not compile successfully).

Once your program compiles successfully, you must restart SAP BC Server to reload the code\libs directory. This makes the service available for execution and allows you to test it with Developer. For details on testing, see “Testing and Debugging Services” on page 247.

**Building Services Using COM**

There are two ways in which you can use COM objects with SAP BC Server. You can:

- **Invoke methods/properties on existing COM or DCOM objects.** SAP BC Server includes built-in services for instantiating any COM or DCOM object registered on your system and invoking its methods and properties. This allows you to use existing COM APIs written in Visual Basic or Visual C++ without writing low-level bridging code. For details, see “Invoking Methods from Existing COM and DCOM Objects” on page 311.

- **Create Services using COM.** SAP BC Server includes libraries for use in your own Visual Basic or Visual C++ code. They allow you to create COM objects that perform work on SAP BC data structures. These objects (compiled into ActiveX DLLs or EXEs) can then be registered as native services, indistinguishable from their Java counterparts.
Requirements

To use SAP BC Server with COM or DCOM, your SAP BC Server must be running Java Virtual Machine 1.2 or later or JView build 5.00.3177 or later (type jview /? from the command prompt to check the build number):

Important! If you modify Visual Basic code intended for use with SAP BC Server libraries, do not use the debug mode in the Visual Basic development environment to test your code. (The debugger does not maintain references to SAP BC Server libraries.) Instead, use a logging feature in your development environment to test the code.

Invoking Methods from Existing COM and DCOM Objects

You can use SAP BC Server to access methods in existing COM and DCOM libraries that do not use SAP BC Values objects. For example, you may have a COM object that performs a validation routine on a String and returns an encrypted String in response. It may not be sensible or desirable to wrap this object with a service if the component is simple enough and/or part of an existing, unmodifiable application. In these cases, the dispatch services can help.

Creating the Object

The win32.COM.dispatch:createObject service (located in the WmWin32 package) will create an object given a Program ID or the Globally Unique Identifier (GUID) for that object. You need to provide this service with the following inputs:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>progid</td>
<td>The program ID of the object that you want to invoke.</td>
</tr>
<tr>
<td>–OR–</td>
<td></td>
</tr>
<tr>
<td>guid</td>
<td>The Globally Unique Identifier (GUID) of the object that you want to invoke.</td>
</tr>
<tr>
<td>context</td>
<td>The context for the object, which is INPROC (DLL), LOCAL_SERVER (EXE), or REMOTE_SERVER (EXE).</td>
</tr>
<tr>
<td>server</td>
<td>DCOM only. The TCP/IP domain name of the machine where the DCOM object is located. For example, doc.rubicon.com or 128.111.222.001.</td>
</tr>
<tr>
<td>user</td>
<td>Optional. DCOM only. The name of the user in which to launch the remote COM object.</td>
</tr>
<tr>
<td>password</td>
<td>Optional. DCOM only. The password associated with user.</td>
</tr>
<tr>
<td>domain</td>
<td>Optional. DCOM only. The Windows domain associated with user.</td>
</tr>
</tbody>
</table>
The service will return a reference to the object called pDispatch or throw an error if the object cannot be created.

Invoking the Object

To invoke methods or properties on this object once you have created it, use win32.COM.dispatch:invoke. You need to supply this service with the following inputs:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDispatch</td>
<td>An object reference previously obtained by the call to createObject, or obtained in the result value of a previous call to invoke.</td>
</tr>
<tr>
<td>dispName</td>
<td>The name of the COM method or property that you want to invoke.</td>
</tr>
<tr>
<td>accessType</td>
<td>Optional. The type of operation (METHOD, GET, PUT, PUTREF) to be performed on dispName. If you are invoking a DCOM object, always set accessType to GET. Incorrect setting of this parameter will cause the invoke to fail. If you are unsure whether a dispName is a method or property, examine the component’s type library using OLEVIEW or a Microsoft development environment.</td>
</tr>
<tr>
<td>params</td>
<td>Optional. An object array of parameters. This is exposed in Developer as an array of Strings for usability (because Objects cannot be manipulated in Developer), but is in reality an Object array. If you need to pass complex or native types, you may have to create this value within your own service.</td>
</tr>
</tbody>
</table>

If the invocation is successful, the return value is contained in “result.” If the result is an object variable, it can then be the target of subsequent calls to invoke by mapping the result to pDispatch in the next invoke.

Writing a Visual Basic Service

Writing services in Visual Basic is easy. To get started quickly, we suggest you examine the sample project file, wmVBDemo.vpb, using Visual Basic 6.0. It resides in <sapbc>\server\packages\WmSamples\code\source.

When you open the project, look at the Services class. It contains one method—a simple “HelloWorld” example. It returns “Hello” added to a name that you pass to the service.

Note: For a complete list of all the methods contained within Values and other objects in the SAP BC COM library, click Object Browser on the View menu in Visual Basic and select the SAP BC library. The DLL (webMethods.dll) is registered and copied into your system directory during installation of Developer and SAP BC Server.
Compiling a Visual Basic Service

To make a Visual Basic class available to SAP BC Server, you need to compile it. To compile the wmVBDemo.vpb example, open the project in Visual Basic and on the File menu, click Make wmVBDemo.dll. You can save the DLL in any directory. (If you are not sure where to put it, <sapbc>/server/packages/WmWin32/code/libs will work.)

Invoking a Visual Basic Service

To execute a service that you have written in Visual Basic, you invoke it using one of SAP BC Server’s COM services. There are two services you can use: the late-binding service (win32.COM:invokeLate) and early-binding service (win32.COM:invoke).

Invoking a VB Service Using Late Binding

Use the following procedure to create a flow service that uses the late-binding service (win32.COM:invokeLate) to invoke a method from a compiled Visual Basic program.

To invoke a VB method using late binding

1. In the Service Browser, select the flow service in which you want to invoke the VB method (if the service does not already exist, use the File ➔ New command to create it).

2. Click ➔ on the Flow Editor toolbar, select Browse and then select win32.COM:invokeLate from the WmWin32 package.

3. Click the Pipeline tab and use the pipeline modifier to assign values to the following variables in Service In:
Click the Input/Output tab and declare the input and output parameters for your service.

If you are invoking the “Hello World” example, declare a String input parameter name and a String output parameter message. For more information about declaring...
input and output parameters, see “To declare input and output parameters for a service” on page 86.

5 Click on the Service Browser toolbar to save the flow service.

6 Click on the Service Browser toolbar to test the service.

Developer prompts you for input values and then displays the results of the service on the Results tab.

**Note:** If you receive an “InvocationTargetException” instead of the results of your service, you may be running an old version of the Microsoft Java Virtual Machine.

**Invoking a VB Service Using Early Binding**

Creating a flow service that uses early binding is very much like creating one that uses late binding. The only difference is that, with early binding, you use `win32.COM:invoke` to invoke the Visual Basic method instead of using `win32.COM:invokeLate`.

Using `win32.COM:invoke` provides better performance. However, it can only be used to call Visual Basic methods that have the following signature:

```vba
Public Sub Sub1(inp as webMethods.Values)
```

**Note:** If you created early-binding services with earlier versions of SAP BC Server (prior to Version 4.0), you will need to update those services and respecify the input parameters for the `win32.COM:invoke` service as described in step 3 of the following procedure.

**To invoke a VB method using early binding**

1 In the Service Browser, select the flow service in which you want to invoke the VB method (if the service does not already exist, use the File > New command to create it).

2 Click on the Flow Editor toolbar, select Browse and then select `win32.COM:invoke` from the WmWin32 package.

3 Click the Pipeline tab, and then use the pipeline modifier to assign values to the following variables in Service In:
Select the Input/Output tab and declare the input and output parameters for your service.

If you are invoking the “Hello World” example, declare a String input parameter name and a String output parameter message. For more information about declaring input and output parameters, see “To declare input and output parameters for a service” on page 86.
5 Click on the Service Browser toolbar to save the flow service.

6 Click on the Service Browser toolbar to test the service.

Developer prompts you for input values, executes the service, and then displays the results of the service on the Results tab.
Creating Client Code

- Basic Concepts ................................................................. 320
- Building a Java Client ....................................................... 320
- Building a C/C++ Client .................................................. 322
- Building a Visual Basic Client ......................................... 324
- Building an Excel Client ................................................ 327
- Building a Browser-Based Client ..................................... 329
CHAPTER 13  Creating Client Code

Basic Concepts

SAP BC Developer enables you to automatically generate client code in a variety of languages and for several environments. Client code is application code that invokes a service on a SAP BC Server. It typically performs the following basic tasks:

- Prompts the user for input values (if your service takes input)
- Places the inputs into an input record (if your service takes input)
- Opens a session on SAP BC Server
- Invokes a service
- Receives output from the service
- Closes a session on SAP BC Server
- Displays the output to the user

The client code that Developer generates can serve as a good starting point for your own development.

Building a Java Client

You can use Developer to generate Java client code that invokes a service.

Assumptions

- For generating the client source code in BC Developer:
  - SAP BC Server is running.
- For compiling the source code on your computer:
  - A fully functional JDK is installed on your computer.
  - Your classpath consists of at least the following:
    `<sapbc>\developer\lib\client.jar`
Third-Party Libraries in client.jar

The following table describes the third-party libraries that SAP BC includes in client.jar.

<table>
<thead>
<tr>
<th>Libraries</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAIK - S/MIME version: 2.6</td>
<td>Institute for Applied Information</td>
</tr>
<tr>
<td>IAIK - iSaSiLk toolkit version: 4.0</td>
<td>Processing and Communications (IAIK)</td>
</tr>
<tr>
<td>IAIK JSSE Wrapper version: 1.2</td>
<td><a href="http://jcewww.iaik.at/">http://jcewww.iaik.at/</a></td>
</tr>
<tr>
<td>Java Naming and Directory Interface™ 1.2.1</td>
<td>Sun</td>
</tr>
<tr>
<td>JavaBeans Activation Framework 1.0.2</td>
<td><a href="http://java.sun.com">http://java.sun.com</a></td>
</tr>
<tr>
<td>International Components for Unicode for Java</td>
<td>IBM</td>
</tr>
<tr>
<td></td>
<td>download/index.html</td>
</tr>
</tbody>
</table>

Limitations

- Developer cannot generate client code for services that use input or output variables that are of type Object or Object List.
- The client code that Developer generates does not support multiple input or output variables with the same name.

If you want to override these limitations, you will need to modify the client code that Developer generates.

Procedure

Use the following procedure to generate Java client code that invokes a service:

To generate Java client code that invokes a service

1. In the Service Browser, select the service for which you want to generate client code.
2. On the Compose menu, click Generate Code.
3. In the Code Generation dialog box, select For calling this service from a client, and then click Next.
4. In the Language field, select Java, and then click Next.
5. Specify the directory where you want Developer to place the generated client code. Either select an existing directory or type the path for a new directory. If you type the path for a new directory, Developer creates the directory.
6 Click Finish.

Developer generates the file that contains the Java client code (ServiceName.java) and a Readme.txt file. The Java client code is written to the hard disk in ISO8859_1, the character set in which the file is encoded.

Modify the generated client code to meet your site’s needs. You can update the client code to invoke built-in services and to use the provided Java API. For information about the built-in services that are available, see the SAP BC Built-In Services Guide. Documentation for the Java API can be found at \<sapbc>\developer\doc\api\index.html.

To complete your client application, refer to the Readme.txt file located in the same directory as your client code.

### Files That Are Generated

This section describes the files that Developer generates for a Java client application.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Readme.txt</td>
<td>File that contains information and instructions for the Java client code. Refer to this file for information about compiling and running the Java client application.</td>
</tr>
<tr>
<td>ServiceName.java</td>
<td>An example file, encoded in ISO8859_1, that contains the application code for the Java client. The application code includes a rudimentary user interface that uses the classes in the FolderName directory. It is not intended for use “as is” in custom applications.</td>
</tr>
</tbody>
</table>

### Building a C/C++ Client

You can use Developer to generate C/C++ client code that invokes a service.

### Assumptions

- SAP BC Server is running.
- A platform that has the C/C++ compiler (e.g., GCC) is installed. SAP generates code for the following platforms: Windows, Solaris, HP-UX, Linux.
- The client.jar file is in the classpath for Developer. The client.jar file is a SAP BC file that is located in the \<sapbc>\developer\lib directory.
- The Make facility is installed.
Limitations

The client code that Developer generates does not support multiple input or output variables with the same name.

If you want to override this limitation, you will need to modify the client code that Developer generates.

Procedure

Use the following procedure to have Developer generate C/C++ client code that invokes a service:

**To generate C/C++ client code that invokes a service**

1. In the Service Browser, select the service for which you want to generate client code.
2. On the Compose menu, click Generate Code.
3. In the Code Generation dialog box, select For calling this service from a client; then click Next.
4. In the Language field, select the C/C++ platform for which you are creating client code. Click Next.
5. Identify the directory where you want Developer to place the generated client code. Either select an existing directory or type the path for a new directory. If you type the path for a new directory, Developer creates the directory.
6. Click Finish.

Developer generates the file that contains the C client code (ServiceName.c), a file that contains compiling settings (ServiceName.make), and a CReadme.txt file.

Modify the generated client code to meet your site's needs. You can update the client code to invoke built-in services and to use the SAP BC provided C API. For information about the built-in services that are available, see the SAP BC Built-In Services Guide. For documentation about the C API, see <sapbc>\developer\doc\api\c\index.html.

To complete your client application, refer to the CReadme.txt file located in the same directory as your client code.

Files That Are Generated

This section describes the files that Developer generates for a C/C++ client application.
Building a Visual Basic Client

You can use Developer to generate Visual Basic client code that invokes a service. Developer creates files that contain the layout and the code for your application.

Assumptions

- SAP BC Server is running.
- Visual Basic Version 5 or 6 is installed.
- SAP BC Type Library 4.0 is installed.

Note: The SAP BC Type Library 4.0 is a COM object that Visual Basic uses to interact with SAP BC Server. The SAP BC Type Library is automatically installed when you install Developer.

Environment Setup

Your system PATH environment variable must include the following directory:

<sapbc>\developer\jvm\bin\hotspot

Limitations

- The client code that Developer generates supports only input values and output values of type String, String List, and String Table.
- The client code that Developer generates does not support multiple input or output variables with the same name.

If you want to override these limitations, you will need to modify the client code that Developer generates.

File Name | Description
--- | ---
CReadme.txt | File that contains information and instructions for the C client code. Refer to this file for information about compiling, running, and deploying your C/C++ client application.

ServiceName.make | A file that contains compiling settings for the C/C++ client. Be sure to update this file with the correct settings for your environment.

ServiceName.c | An example file that contains the C/C++ client code. It is not intended for use as-is in custom applications.

Note: The SAP BC Type Library 4.0 is a COM object that Visual Basic uses to interact with SAP BC Server. The SAP BC Type Library is automatically installed when you install Developer.
Procedure

Use the following procedure to have Developer generate Visual Basic client code that invokes a service.

To generate Visual Basic client code that invokes a service

1. In the Service Browser, select the service for which you want to generate client code.
2. On the Compose menu, click Generate Code.
3. In the Code Generation dialog box, select For calling this service from a client, and click Next.
4. In the Language field, select Visual Basic 5/6, and click Next.
5. Identify the directory where you want Developer to place the generated client code. Either select an existing directory or type the path for a new directory. If you type the path for a new directory, Developer creates the directory.
6. Click Finish.

Developer generates several files, including the serviceNameReadMe.txt file. This file contains detailed information about all the generated files. Refer to it to complete your client application and for information about deploying your client application.

Files That Are Generated

This section describes the files that Developer generates for a Visual Basic client application.

General Files

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceName.vbp</td>
<td>The Visual Basic project file.</td>
</tr>
<tr>
<td>ServiceNameReadme.txt</td>
<td>The file that contains information and instructions for the Visual Basic client code. Refer to this file for information about deploying your Visual Basic client application.</td>
</tr>
</tbody>
</table>
### Files for the User Interface

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>frmArrayInput.frm</td>
<td>Contains layout and code that is used if any of the input values for the service are of type String List.</td>
</tr>
<tr>
<td>frmOutput.frm</td>
<td>Contains layout and code that is used when the service returns output. It also contains the Setup, Invoke, and Exit buttons.</td>
</tr>
<tr>
<td>frmSetup.frm</td>
<td>Contains layout and code that prompts for the server properties for the SAP BC Server on which the service to execute resides.</td>
</tr>
<tr>
<td>frmStringInput.frm</td>
<td>Contains layout and code that is used if any of the input values for the service are of type String.</td>
</tr>
<tr>
<td>frmTableInput.frm</td>
<td>Contains layout and code that is used if any of the input values for the service are of type String Table.</td>
</tr>
<tr>
<td>wmSampleLib.bas</td>
<td>Contains code that is specific to the sample template that Developer generates.</td>
</tr>
</tbody>
</table>

### Files Containing the Code that Invokes the Service

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceName.bas</td>
<td>An example file that illustrates how to use the service class in an application. This module is dependent on objects in the project template that SAP provides. It is not intended for use “as is” in custom applications.</td>
</tr>
<tr>
<td>ServiceName.cls</td>
<td>The service object. You include this object in your own project.</td>
</tr>
</tbody>
</table>

### File Containing the Code that Interacts with SAP BC Server

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wmVBConnection.bas</td>
<td>Code used as a layer of abstraction to interact with SAP BC Server.</td>
</tr>
</tbody>
</table>
Building an Excel Client

You can use Developer to generate client code that executes a service from a MS Excel spreadsheet.

Assumptions

- SAP BC Server is running.
- Excel 97 or Excel 2000 is installed.
- SAP BC Type Library 4.0 is installed.

**Note:** The SAP BC Type Library 4.0 is a COM object that Visual Basic uses to interact with SAP BC Server. The SAP BC Type Library is automatically installed when you install Developer.

Limitations

- The client code that Developer generates only supports input values of type String and output values of type String, String List, and String Table.
- The client code that Developer generates does not support multiple input or output variables with the same name.

If you want to override these limitations, you will need to modify the client code that Developer generates.
**Procedure**

Use the following procedure to have Developer generate Excel client code that invokes a service.

1. In the Service Browser, select the service for which you want to generate client code.
2. On the Compose menu, click Generate Code.
3. In the Code Generation dialog box, select For calling this service from a client, and click Next.
4. In the Language field, select Excel 97/2000, and click Next.
5. Identify the directory where you want Developer to place the generated client code. Either select an existing directory or type the path for a new directory. If you type the path for a new directory, Developer creates the directory.
6. Click Finish.

   Developer generates several files, including the serviceNameReadMe.txt file. This file contains detailed information about all generated files.

7. Copy the wmXLTemplate.xls file, which SAP provides, to the directory that contains the client code that Developer generated. The wmXLTemplate.xls file is located in the <sapbc>\developer\support\Excel directory.

8. Open the wmXLTemplate.xls file. When prompted to indicate whether you want to enable macros, select Enable Macros.

9. Follow the instructions in the wmXLTemplate.xls file to complete your client application. See the serviceNameReadMe.txt file for information about deploying your client application.
Files That Are Generated

This section describes the files that Developer generates for an Excel client application.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceName.Readme.txt</td>
<td>A file that contains information and instructions for the Visual Basic client code. Refer to this file for information about deploying your Visual Basic client application.</td>
</tr>
<tr>
<td>ServiceName.bas</td>
<td>An example file that illustrates how to use the service class in a spreadsheet. This module is dependent on objects in the project template that SAP provides. It is not intended for use “as is” in custom applications.</td>
</tr>
<tr>
<td>ServiceName.cls</td>
<td>The service object. You include this object into your own project.</td>
</tr>
</tbody>
</table>

Building a Browser-Based Client

You can invoke a service with a URL. This means that you can invoke a service by entering the URL into your Web browser or by embedding the URL in Web pages.

To build a browser-based client, create one or more Web pages that invoke URLs for one or more services. When SAP BC Server receives the first URL from a Web browser, it creates a session for the client on SAP BC Server. The session information is stored in a cookie in the browser. As the user of the browser-based application clicks on links to URLs that invoke services, SAP BC Server uses the cookies to find session information for the client. SAP BC Server keeps the session information for the client until the session expires. Sessions expire based on the configured session timeout value. For more information about setting the session timeout limit, refer to the SAP BC Administration Guide.

**Note:** You cannot use Developer to generate browser-based clients.

Assumptions

- SAP BC Server is running.
- The input values for the services you want to invoke are determined. You will need to include the input values in the URL that you use to invoke a service.
Limitations

When you test a service using the Run in Browser command, only input values of the type String and String List will be passed to the service. Input values of the type Record, Record List, Object, and Object List will not be displayed when the Web page is served.

Invoking Services with a URL

First, build your Web pages using any tool you choose. To invoke the URL, use either the HTTP GET or the POST method. In either case, use a URL similar to the following:

```
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Identifies the SAP BC Server on which the service you want to invoke resides.</td>
</tr>
<tr>
<td>2</td>
<td>Specifies the required keyword “invoke”, which tells SAP BC Server that the URL identifies a service that is to be invoked.</td>
</tr>
<tr>
<td>3</td>
<td>Identifies the folder in which the service to invoke resides. Separate subfolders with periods. This field is case-sensitive. Be sure to use the same combination of upper and lower case letters as specified in the folder name on SAP BC Server.</td>
</tr>
<tr>
<td>4</td>
<td>Identifies the service that you want to invoke. This field is case-sensitive. Be sure to use the same combination of upper and lower case letters as specified in the service name on SAP BC Server.</td>
</tr>
<tr>
<td>5</td>
<td>Specifies the input values for the service. Specify a question mark (?) before the input values. The question mark signals the beginning of input values. Each input value is represented as variable=value. The variable portion is case-sensitive. Be sure to use the same combination of upper and lower case letters as specified in your service. If your service requires more than one input value, separate each variable=value with an ampersand (&amp;).</td>
</tr>
</tbody>
</table>

**Note:** Only specify this part of the URL when using the HTTP GET method.

**Note:** If you are serving the Web pages that invoke services from a SAP BC Server, you can use a relative URL to invoke the service. By doing so, you can serve the exact Web page from several servers without having to update the URLs.
Using the HTTP GET Method

To use the GET method, embed a URL that includes all the input values for the service in the query string portion of the URL. When the server receives the URL, it translates the input values into an IData object. For more information about how the server creates the IData object that it sends to the service, refer to “Input into the Service” on page 331.

Using the HTTP POST Method

To use the POST method, create an HTML form in your Web page. Create fields in the HTML form in which a user will supply the input information. The values you specify for the NAME attributes of the HTML form fields should match the names of input values that the service expects. Be sure to use the exact combination of upper and lower case letters as specified in your service. For example, if your service requires the input values `sku` and `quantity`, you might create an HTML form with the following fields:

```html
<SELECT NAME="sku">
  <OPTION VALUE="A1">A1</OPTION>
  <OPTION VALUE="B2">B2</OPTION>
  <OPTION VALUE="C3">C3</OPTION>
</SELECT>

(INPUT TYPE="TEXT" NAME="quantity" VALUE="1")
```

Specify the URL for the service in the ACTION attribute and “POST” in the METHOD attribute. For example:

```html
<FORM ACTION="/invoke/sample.webPageDemo/getProductCost" METHOD="POST">
```

After the user fills in the form and submits it, the Web browser creates a document that contains the information the user supplied in the HTML form (performs an HTTP POST). The browser invokes the URL identified in the ACTION attribute, which invokes the service on SAP BC Server, and the browser posts the document that contains the user’s input information to SAP BC Server. For more information about how the server creates the IData object that it sends to the service, see “Input into the Service” on page 331.

Input into the Service

Regardless of whether SAP BC Server receives a URL that uses the HTTP GET or POST method, it creates an IData object from the input information. It then passes the IData object to the specified service. This becomes the pipeline upon which the service operates.

To create the pipeline object, SAP BC Server creates two key/value pairs for each input value—one of type String and one of type String List. For example, if the input values contain the variable `sku` with value `A1` and `quantity` with value `1`, the service is passed the following IData object:
When the server receives multiple input values that are associated with the same variable name, the String variable in the IData object contains only the value of the first variable; the String List variable contains all values. For example, the following shows a URL that contains two values for the variable year and the resulting IData object that the server creates:

```
```

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>1998</td>
<td>String</td>
</tr>
<tr>
<td>yearList</td>
<td>1998</td>
<td>String List</td>
</tr>
<tr>
<td></td>
<td>1999</td>
<td></td>
</tr>
</tbody>
</table>

Similarly, if the HTML form contains two fields with the same name and a user supplies values for more than one, the String variable in the IData object contains only the value of the first variable; the String List variable contains all values. For example, the following shows sample HTML code that renders check boxes:

```
<input type="checkbox" name="Color" value="blue">Blue<br>
<input type="checkbox" name="Color" value="green">Green<br>
<input type="checkbox" name="Color" value="red">Red<br>
```

If the browser user selects all check boxes, the document that is posted to SAP BC Server will contain three values for the variable named Color. The following shows the IData object that the server passes to the service:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
<td>blue</td>
<td>String</td>
</tr>
<tr>
<td>ColorList</td>
<td>blue</td>
<td>String List</td>
</tr>
<tr>
<td></td>
<td>green</td>
<td></td>
</tr>
<tr>
<td></td>
<td>red</td>
<td></td>
</tr>
</tbody>
</table>
Output from the Service

By default, SAP BC Server displays the output from a service in a HTML Web page, using a table to render the output values. However, you can format the output using output templates. You can use an output template that formats the output from one service and includes a URL that invokes another service. For more information about output templates, see “Assigning an Output Template to a Service” on page 91.
Working with WSDL

- Overview ................................................................. 336
- What Is WSDL? ......................................................... 336
- What Does a WSDL Document Look Like? ................... 336
- Generating a WSDL Document for a Service ................. 340
- Creating a Web Service Connector ............................. 366
Overview

The SAP BC platform supports WSDL (Web Services Description Language) by providing features that enable you to create a WSDL document from any service and create a service (a Web Service Connector) from a valid WSDL document. A Web Service Connector is a flow service that implements client connection information to invoke a Web Service located on a remote server.

What Is WSDL?

Web Services Description Language (WSDL) is an XML format for describing Web Services accessible through the Internet. A WSDL document specifies:

- What work the Web Service performs.
- Where the Web Service is located.
- How to access the Web Service, including the protocol used to access the Web Service and the protocol used to return a response.
- What data is exchanged by the Web Service, specifically the input parameters supplied to the Web Service and the output parameters produced by the Web Service.

A Web Service provider can create a WSDL document for the service, upload the WSDL document to a publicly accessible server, and use a Web Services registry to publish a URL that points to the WSDL document location. Web Service consumers can browse the registry, discover the service, and then use the supplied URL to download the WSDL document. The WSDL document contains all of the information the consumer needs to create a proxy service that sends data to the Web Service, invokes the Web Service, and receives data from the Web Service.

What Does a WSDL Document Look Like?

A WSDL document uses seven basic elements to describe where the Web Service is located, how to access it, and the signature of the service. Any valid WSDL document may contain the following basic elements:
### What Does a WSDL Document Look Like?

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;types&gt;</td>
<td>Contains the type definitions that describe the data exchanged by the Web Service. The &lt;types&gt; element can reference entire XML Schemas and can contain simple type definitions, complex type definitions, and element declarations. The type definitions and element declarations help define the input and output parameters for the Web Service. WSDL uses XML Schema as its native type system.</td>
</tr>
<tr>
<td>&lt;message&gt;</td>
<td>Specifies the data being exchanged by the Web Service. A &lt;message&gt; element describes a set of input parameters or a set of output parameters. Each &lt;message&gt; element can contain one or more &lt;part&gt; elements. A &lt;part&gt; element associates a piece of data with a name and a type definition or element declaration. The type definition or element declaration referenced by the &lt;part&gt; element can be defined, declared, or referenced in the &lt;types&gt; element.</td>
</tr>
<tr>
<td>&lt;operation&gt;</td>
<td>Specifies the messages received and sent by the Web Service. Within the &lt;operation&gt; element, the &lt;input&gt; element identifies the message whose parts specify the input parameters to the Web Service, and the &lt;output&gt; element identifies the message whose parts specify the output parameters of the Web Service. Essentially, the operation specifies the signature for the Web Service. An &lt;operation&gt; element is declared within a &lt;portType&gt; element.</td>
</tr>
<tr>
<td>&lt;portType&gt;</td>
<td>Defines a named set of operations. The &lt;portType&gt; element associates a port type name with a set of operations. A &lt;portType&gt; element can contain multiple operations.</td>
</tr>
<tr>
<td>&lt;binding&gt;</td>
<td>Specifies the protocol and message format to use to access the operations in a port type. Each &lt;binding&gt; element can specify only one protocol for a port type. However, a WSDL document can define more than one binding for a single port type. A WSDL document should include one &lt;binding&gt; element for each protocol that it supports.</td>
</tr>
<tr>
<td>&lt;port&gt;</td>
<td>Associates a binding with a network address. Together, the binding and network address specify how to invoke a Web Service. Each port can specify only one network address for a binding. However, multiple ports can be defined for a single Web Service. Port elements are defined within the &lt;service&gt; element.</td>
</tr>
<tr>
<td>&lt;service&gt;</td>
<td>Identifies all of the ports that can be used to call a Web Service. A &lt;service&gt; element can contain many ports. Each port in a &lt;service&gt; element represents an alternative way of calling the same Web Service. The ports access the same Web Service (refer to the same port type), but use different bindings (protocols) or network address to invoke the Web Service.</td>
</tr>
</tbody>
</table>
In a WSDL document, the elements that describe a Web Service are enclosed in the `<definitions>` element.

### A Sample WSDL Document

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="AuthenticateUser"
    targetNamespace="http://www.example.com"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/
    xmlns:tns="http://www.example.com"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/
    xmlns:xsd1="http://example.com/AuthenticateUser.xsd">
    <wsdl:types>
        <xsd:schema targetNamespace="http://example.com/AuthenticateUser.xsd"
            elementFormDefault="qualified">
            <xsd:complexType name="AuthenticateUserInput">
                <xsd:sequence>
                    <xsd:element name="userName" type="xsd:string"/>
                    <xsd:element name="password" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="AuthenticateUserOutput">
                <xsd:sequence>
                    <xsd:element name="isValid" type="xsd:boolean"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:schema>
    </wsdl:types>

    <wsdl:message name="AuthenticateUserInput">
        <wsdl:part name="composite" type="xsd1:AuthenticateUserInput"/>
    </wsdl:message>
    <wsdl:message name="AuthenticateUserOutput">
        <wsdl:part name="composite" type="xsd1:AuthenticateUserOutput"/>
    </wsdl:message>

    <wsdl:portType name="AuthenticateUserPortType">
        <wsdl:operation name="AuthenticateUser">
            <wsdl:input name="AuthenticateUserInput" message="tns:AuthenticateUserInput"/>
            <wsdl:output name="AuthenticateUserOutput" message="tns:AuthenticateUserOutput"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```
What Does a WSDL Document Look Like?

The WSDL Namespace Declaration

A WSDL document uses the namespace http://schemas.xmlsoap.org/wsd/ to qualify the elements and attributes used to create the framework of a WSDL document. By convention, the prefix wsdl is given to the WSDL namespace. The primary purpose of the WSDL namespace is to distinguish WSDL-related elements and attributes from elements and attributes defined in other namespaces.

A WSDL document also uses namespaces such as http://schemas.xmlsoap.org/wsd/soap/ to indicate that an element or attribute belongs to the WSDL namespace for a WSDL SOAP binding, or http://schemas.xmlsoap.org/wsd/http/ to indicate that an element or attribute is defined in the WSDL namespace for an HTTP GET and POST binding. The following namespace prefixes are commonly found in a WSDL document and are inserted in WSDL documents generated by the WSDL generator. (The WSDL generator is the subsystem of the SAP BC Server that creates a WSDL document for a service.)
Generating a WSDL Document for a Service

Any service on a SAP BC Server can be made available to external clients by generating a WSDL document for the service. The WSDL document and its supporting files can be distributed to clients (Web Service consumers) or made available for consumers to locate and download from a publicly accessible web server. The subsystem of the SAP BC Server that creates WSDL documents is called the WSDL generator.

When you generate a WSDL document for a service, you need to provide the following information:

- **Location.** Specify the host name and port number of the SAP BC Server on which the service is located. If you are generating a WSDL for a service currently located on a development server and you intend to distribute the document to partners, make sure to specify the host name and port number for the production server.

- **Protocol.** Specify the protocol that Web Service that consumers need to use to communicate with the service. You can specify the following protocols: SOAP RPC, SOAP message (document), HTTP POST, and HTTP GET.

- **Transport.** Specify the transport Web Service consumers need to use to invoke the service. You can specify HTTP or HTTPS.

### Prefix | Namespace URI | Description
---|---|---
soap | http://schemas.xmlsoap.org/wsdl/soap/ | Namespace defined by the WSDL specification for associating a binding with the SOAP protocol.
mime | http://schemas.xmlsoap.org/wsdl/mime/ | Namespace defined by the WSDL specification for associating a binding with the MIME protocol.
http | http://schemas.xmlsoap.org/wsdl/http/ | Namespace defined by the WSDL specification for associating a binding with the HTTP GET & POST protocol.
Generating a WSDL Document for a Service

- **Service Signature.** Specify the input and output parameters that the service expects and produces. For some protocols, such as RPC, the WSDL generator can use the parameters declared on the **Input/Output** tab to create the service signature. For other protocols, such as SOAP message and HTTP POST, you need to select a record or XML Schema component to describe the input and/or output signature of the service. The record or XML Schema component describes the XML document the service expects as input and the XML document the service produces as output.

- **Target Namespace.** Specify the namespace to which the elements declared in the generated WSDL document belong.

When the WSDL generator creates a WSDL document, it uses the information you provide along with information from the service to build the WSDL document and the supporting XML Schema definition files.

The following sections provide more information about generating a WSDL document for each protocol.

### Generating a WSDL Document that Uses the SOAP RPC Protocol

When you specify SOAP-RPC as the protocol, the RPC processor built into SAP BC Server will receive SOAP messages that invoke the service. The RPC processor is the subsystem that SAP BC Server uses to receive, process, and send remote procedure calls. For more information about SOAP, see the **SAP BC SOAP Programming Guide**.

#### To generate a WSDL document that uses SOAP RPC protocol

1. In the Service Browser, select the service for which you want to generate a WSDL document.
2. On the **Compose** menu, select **Generate WSD**. Developer opens the **Generate WSD** dialog box.
Generate WSD dialog box

Developer populates the **Namespace name** and **Local name** fields only if you assigned an explicit universal name to the selected service. For information about assigning universal names to services, see the SAP BC SOAP Programming Guide.

3 In the **Host** field, type the numeric IP address or the domain name of the SAP BC Server on which the service will reside. You do not need to specify http:// or https:// as part of the host name. The WSDL generator automatically adds http://or https:// when it compiles the network address for the service.

In the **Host** field, Developer automatically inserts the name of the server on which the service currently resides.

**Note:** If you are generating a WSDL document for a service located on your development server and you intend to distribute the document to partners, make sure that you enter the domain name or IP address of the production server in the **Host** field.

4 In the **Port** field, type the number of a port on the host server that you want to accept requests for this service. The port needs to accept HTTP or HTTPS requests.

In the **Port** field, Developer automatically inserts the port number you used to open the current session on SAP BC Server.

5 Under **Protocol**, select **SOAP-RPC**.

Developer automatically enters “rpc” in the **Directive** field. When you specify SOAP-RPC as the protocol, the built-in SOAP RPC processor receives, processes, and responds to remote procedure calls for the service.
6 Under **Via Transport**, do one of the following:

<table>
<thead>
<tr>
<th>Select...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>Specify that requests to invoke the service must be sent via HTTP.</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Specify that requests to invoke the service must be sent securely using HTTPS.</td>
</tr>
</tbody>
</table>

**Note:** The transport you select must be the same as the type of requests that the port you selected accepts. If the port accepts HTTP requests, select **HTTP** under **Transport**. If the port accepts HTTPS requests, select **HTTPS**.

7 In the **Target Namespace** field, type the URI you want to use as the target namespace for the WSDL document. In the generated WSDL document, the elements, attributes, and type definitions will belong to this namespace. By default, Developer displays the following as the target namespace:

http://host

where **host** is the name of the SAP BC Server to which you are currently connected.

**Note:** Even if you delete the target namespace that Developer automatically enters, the WSDL generator will use **http://host** as the target namespaces in the WSDL document. If you want a null target namespace, you will need to edit the generated WSDL document.

8 Click **OK**. Developer displays the **Generate WSD** dialog box.

9 Under **Select Directory for Saving WSD** files, enter the directory to which you want to save the generated WSDL document and other generated files. Either select or enter an existing directory or type the path for a new directory. If you type the path for a new directory, the WSDL generator creates the directory when it generates the files.

10 Click **OK**.

When the WSDL generator finishes generating the WSDL document and the XML Schema definitions, SAP BC Developer displays a message listing the files that were generated and the directory in which the files were placed.

If an error occurs during WSDL generation, SAP BC Developer displays a message stating that an error occurred. If the WSDL generator created files before the error occurred, SAP BC Developer displays a message listing the files that were generated and the directory in which the files were placed.

**Note:** If files of the same name exist already in the selected directory, SAP BC Developer displays a warning stating that files will be overwritten.
## What Files are Generated?

The WSDL generator creates the following files when it creates a WSDL document to describe a service.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>localName.wsdl</td>
<td>WSDL document that contains a description of the service, including how to invoke the service, where the service is located, the input parameters required by the service, and the output parameters produced by the service. For the SOAP protocols, the WSDL generator uses the local name component of the universal name to name the WSDL document. For the HTTP protocols, the WSDL generator uses the local name component of the implicit universal name to name the WSDL document. For more information about how the WSDL generator uses the universal name to name elements in the WSDL document, see “About the Generated WSDL Document” on page 345. The WSDL generator always generates this file.</td>
</tr>
<tr>
<td>default.xsd</td>
<td>XML Schema definition containing the type definitions and element declarations that represent the parameters in the input and output signature of the service. This XML Schema definition does not have target namespace.</td>
</tr>
<tr>
<td>prefix.xsd</td>
<td>XML Schema definition containing type definitions and element declarations for prefixed variables in the input and output signature. The WSDL generator creates an XML Schema definition for each unique prefix. The prefix corresponds to the namespace in which the type definitions and element declarations belong.</td>
</tr>
</tbody>
</table>
Generating a WSDL Document for a Service

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST#.xsd</td>
<td>XML Schema definition containing the type definitions for the Content type properties applied to parameters declared in a record or on the Input/Output tab. A content type corresponds to a simple type definition in an SAP BC schema (ST=Simple Type). If the record or Input/Output tab contains variables with Content type properties from an SAP BC schema other than pub.schema.w3c:datatypes, the WSDL generator creates an XML Schema for each unique namespace. The first XML Schema is named ST1.xsd, the second is named ST2.xsd.</td>
</tr>
<tr>
<td>namespaces.dtd</td>
<td>A DTD (Document Type Definition) that lists each prefix used in the service signature and its corresponding namespace. These namespace and prefix declarations are used by all the XML Schema definition files. The WSDL generator only generates this file when more than one XML Schema definition is generated.</td>
</tr>
</tbody>
</table>

**Note:** Because the WSDL generator uses ST followed by a number as the naming convention for generated XML Schemas, avoid using ST# as the prefix in a variable name, i.e., ST2:myType in records.

**Important!** The generated files use `<include>` elements to reference the contents of other generated files. The `<include>` element uses a relative path to specify the location of the other files. For example, the `<types>` element in the WSDL document contains an `<include>` element that uses `schemaLocation="default.xsd"` to reference the default.xsd. The location is a relative path. If you move the WSDL document or the default.xsd or rename default.xsd so that the relative path changes, make sure to update the `<include>` elements with the new relative path or the new absolute path.

**About the Generated WSDL Document**

The WSDL generator in SAP BC Server uses the host, protocol and transport information you select along with the service information to generate a WSDL document. For the SOAP protocols, the WSDL generator also uses the universal name of the service to name the elements in the WSDL document. A **universal name** is a unique public identifier that external protocols use to reference a service on SAP BC Server. A universal name consists of two parts: a namespace name and a local name. The WSDL generator uses either the namespace name or the local name as the name of the WSDL element. (Specifically, the WSDL generator uses the namespace name or the local name as the value of the `name` attribute in a WSDL element.)
**Note:** Each service can have an explicit universal name and an implicit universal name. You may optionally assign an explicit universal name to the service. Every service that exists on SAP BC Server has an implicit universal name. An implicit universal name is derived from the name of the service itself. For implicit universal names, the namespace name is the fully qualified name of the folder in which the service resides. The local name is the unqualified name of the service. For more information about universal names, see “Assigning Universal Names to Services” on page 98 and the SAP BC SOAP Programming Guide.

For the HTTP protocols, the WSDL generator uses the namespace name and local name components of the implicit universal name as the value of the `name` attributes for WSDL elements.

The following table identifies the naming conventions the WSDL generator uses to assign a value to the `name` attributes for WSDL elements.

<table>
<thead>
<tr>
<th>This WSDL element...</th>
<th>Is assigned this name...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;definitions&gt;</code></td>
<td><code>namespaceName</code></td>
</tr>
<tr>
<td><code>&lt;message&gt;</code> (first)</td>
<td><code>localNameInput</code></td>
</tr>
<tr>
<td><code>&lt;part&gt;</code></td>
<td><code>composite</code></td>
</tr>
<tr>
<td>(in first <code>&lt;message&gt;</code> element)</td>
<td>The WSDL generator assigns the <code>type</code> attribute the value <code>localNameInput</code>.</td>
</tr>
</tbody>
</table>

**Note:** The WSDL generator uses `localNameInput` as the value of the `type` attribute. When you specify SOAP-MSG or HTTP-POST as the protocol, you can select a type definition or element declaration in an XML Schema to describe the input signature. If you specified a type definition as the input signature, the WSDL generator uses the type definition name as the value of the `type` attribute. If you specified an element declaration as the input signature, the `<part>` element carries the `element` attribute and the WSDL generator uses the element name as the value of the `element` attribute.
Generating a WSDL Document for a Service

This WSDL element... Is assigned this name...

<message> (second) localNameOutput

If the service from which you create the WSDL document does not declare any output parameters, or you do not select a record or XML Schema component to describe the service output, the WSDL generator does the following:

- For SOAP-RPC and SOAP-MSG protocols, the WSDL generator inserts an empty output <message> element into the WSDL document.

- For HTTP-POST and HTTP-GET protocols, the WSDL generator does not create an output <message> element.

For SOAP-RPC and SOAP-MSG protocols, the WSDL generator inserts an empty output <message> element into the WSDL document.

For HTTP-POST and HTTP-GET protocols, the WSDL generator does not create an output <message> element.

Note: Part names need to be unique only within a <message> element. Different <message> elements can have parts of the same name.

This WSDL element... Is assigned this name...

<part> composite

The WSDL generator assigns the type attribute the value localNameOutput.

Note: Part names need to be unique only within a <message> element. Different <message> elements can have parts of the same name.

This WSDL element... Is assigned this name...

<portType> namespaceNamePortType

<operation> localName

<input> localNameInput

<output> localNameOutput

<binding> namespaceNameBinding

<service> namespaceNameService

<port> namespaceNamePort0
About the Address Element

Within the `<port>` element in a WSDL document, the `<address>` element specifies the location of the Web Service. More specifically, the `<address>` element carries a location attribute that specifies the network address for the service. The WSDL generator creates a value for this attribute using the protocol, transport, host, port, and directive information that you provide in the Generate WSD dialog box.

For WSDL documents that specify SOAP-RPC or SOAP-MSG as the protocol, the location attribute in the `<address>` element has the following format:

transport://host:port/soap/directive

For WSDL documents that specify HTTP-POST or HTTP-GET as the protocol, the location attribute in the `<address>` element is used in combination with the location attribute in the `<operation>` element to specify the URL used to invoke the service. For WSDL documents that specify HTTP-POST and HTTP-GET as the protocol, the location attribute in the `<address>` element has the following format:

transport://host:port/directive

where directive is almost always “invoke”.

The location attribute in the `<operation>` element specifies the fully qualified path name of the service on the SAP BC Server and has the following format:

folder/subfolder/serviceName

About the Generated XML Schema Definitions

When the WSDL generator creates a WSDL document, it may also generate one or more XML Schema definitions. The XML Schema definition contains the type definitions and element declarations that correspond to the parameters in the input and output signature of the service.

The XML Schema definitions contain a global element declaration for each input and output parameter declared in the service signature. The default.xsd file also contains a global complex type definition for the input parameters and a global complex type definition for the output parameters. These complex type definitions are named localNameInput and localNameOutput, respectively. (An exception to this is that the input complex type definition for a service accessible via the SOAP-MSG protocol is named localName.)

The localNameInput and localNameOutput complex type definitions specify a `<sequence>` element that contains a local element declaration for each top-level parameter (variable) in the input or output signature. Each local element declaration in these complex type definitions is actually an element reference to the global element declarations created for each top-level parameter. Within the element reference, the values of the minOccurs and maxOccurs attributes specify whether the parameter is optional or required, and whether the parameter is an array.
If the element reference contains `minOccurs="0"`, then the element corresponds to a optional parameter.

If the element reference contains `minOccurs="1"`, then the element corresponds to a required parameter.

If the element reference contains `maxOccurs="unbounded"`, then the parameter corresponds to a one-dimensional array variable (String List, Object List, Record List, or Record Reference List).

The `<sequence>` elements in the `localNameInput` and `localNameOutput` complex type definitions also contain an `<any>` element declaration. This declaration allows the input XML document for the service to contain undeclared fields.

Each top-level parameter declared in the input and output signature corresponds to a global element declaration in the XML schema. The type declaration for an element depends on the parameter data type.

- If the parameter is a String or a String List, the type definition for the element is determined by the `Content type` property. For example:
  - If the `Content type` is a built-in data type, such as `decimal` (http://www.w3.org/2001/XMLSchema), then the element declaration contains `type="xsd:decimal"`.
  - If the `Content type` is a customized type or a simple type defined in an SAP BC schema, then the element declaration contains `type="QName"`. If the `Content type` is namespace qualified (a simple type defined in an SAP BC schema with a target namespace), the QName contains a prefix in the format ST#, such as ST1 or ST2. The simple type definition for the `Content type` appears in the corresponding XML Schema generated for the SAP BC schema (named ST#). If the `Content type` is not namespace qualified (a customized content type), the default.xsd contains a simple type definition for the `Content type`.
  - If the `Content type` is not specified, the parameter is treated as if the `Content type` property were set to `string` (http://www.w3.org/2001/XMLSchema). The element declaration contains `type="xsd:string"`.

- If the parameter is a Record or Record List, the corresponding element is defined to be of anonymous complex type. The in-line, anonymous complex type definition contains a `<sequence>` with local element declarations that correspond to each child parameter in the Record or Record List variable. If the Record or Record List variable allowed unspecified fields to exist at run time, the `<sequence>` contains an `<any>` element declaration.
If the parameter is a Record Reference or a Record Reference List, the corresponding element is defined to be of complex type. The complex type is assigned the fully qualified name of the referenced record. (That is, the element declaration contains type="folderName.subfolderName-recordName"). The XML Schema definition then contains a global complex type definition for the referenced record. If the Record Reference or Record Reference List variable allowed unspecified fields to exist at run time, the <sequence> contains an <any> element declaration.

Note: The above is true only for top-level Record Reference and Record Reference List parameters. Elements that correspond to Record Reference and Record Reference List parameters that are children of other parameters are defined to be of anonymous complex type.

If the parameter is an Object or Object List, the corresponding element is determined to be an anyType [http://www.w3.org/2001/XMLSchema]. The element declaration for the parameter contains type="xsd:anyType".

If the parameter is a String Table, the corresponding element is defined to be of complex type. This complex type is named variableName_itemType. The XML Schema contains a global complex type definition for variableName_itemType which is expressed as a SOAP array.

Note: Whenever the input or output signature for a service includes a String Table, all multi-dimensional variables are expressed as SOAP arrays.

Note: The WSDL generator does not support records that contain identically named variables at the top level or records with recursive references (a record that references itself). If you generate a WSDL document from a service that uses these unsupported record structures, the resulting XML Schema definition or WSDL document may be invalid.

The following table summarizes how the SAP BC data types correspond to components in an XML Schema.
### Generating a WSDL Document for a Service

<table>
<thead>
<tr>
<th>This SAP BC Data Type...</th>
<th>Is represented as this in an XML Schema...</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>An element of simple type. The type for the element is determined by the <strong>Content type</strong> property.</td>
</tr>
<tr>
<td></td>
<td>- If the <strong>Content type</strong> is a built-in data type, such as <code>decimal</code> [<a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a>], then the element declaration contains <code>type=&quot;xsd:decimal&quot;</code>.</td>
</tr>
<tr>
<td></td>
<td>- If the <strong>Content type</strong> corresponds to a customized type, the default.xsd contains a simple type definition for the customized type.</td>
</tr>
<tr>
<td></td>
<td>- If the <strong>Content type</strong> corresponds to a user-defined type in an SAP BC schema with a target namespace, the XML Schema generated for that namespace contains the simple type definition for the user-defined type.</td>
</tr>
<tr>
<td></td>
<td>- If no <strong>Content type</strong> is set, the element is of type <code>xsd:string</code>.</td>
</tr>
<tr>
<td>String List</td>
<td>An element of simple type. The type for the element is determined by the <strong>Content type</strong> property.</td>
</tr>
<tr>
<td></td>
<td>- If the <strong>Content type</strong> is a built-in data type, such as <code>decimal</code> [<a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a>], then the element declaration contains <code>type=&quot;xsd:decimal&quot;</code>.</td>
</tr>
<tr>
<td></td>
<td>- If the <strong>Content type</strong> corresponds to a customized type, the default.xsd contains a simple type definition for the customized type.</td>
</tr>
<tr>
<td></td>
<td>- If the <strong>Content type</strong> corresponds to a user-defined type in an SAP BC schema with a target namespace, the XML Schema generated for that namespace contains the simple type definition for the user-defined type.</td>
</tr>
<tr>
<td></td>
<td>- If no <strong>Content type</strong> is set, the element is of type <code>xsd:string</code>.</td>
</tr>
<tr>
<td>String Table</td>
<td>An element of complex type, where the complex type is expressed as a SOAP array.</td>
</tr>
<tr>
<td>Record</td>
<td>An element of anonymous complex type. The complex type definition contains a <code>&lt;sequence&gt;</code> with a local element declaration for every child parameter in the Record.</td>
</tr>
</tbody>
</table>

When a global element declaration corresponding to a String List parameter is referenced from a `<sequence>`, the `maxOccurs` attribute is set to "unbounded".
Generating a WSDL that Uses the SOAP Message Protocol

When you specify SOAP messaging as the protocol for invoking the service described by the WSDL document, you can select the SOAP processor (the directive) that you want to handle the message sent by consumers of the Web Service. You can select either the default processor or a registered, custom SOAP processor that you built to suit your needs. The processor you select will receive, process, and send SOAP messages that invoke the service.

The service for which you are generating the WSDL document will be a target service of the selected processor. Consequently, the service needs to conform to the target service requirements for the specified processor.

<table>
<thead>
<tr>
<th>This SAP BC Data Type...</th>
<th>Is represented as this in an XML Schema...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record List</td>
<td>An element of anonymous complex type. The complex type definition contains a <code>&lt;sequence&gt;</code> with a local element declaration for every child parameter in the Record List. When a global element declaration corresponding to a Record List parameter is referenced from a <code>&lt;sequence&gt;</code>, the maxOccurs attribute is set to &quot;unbounded&quot;.</td>
</tr>
<tr>
<td>Record Reference</td>
<td>An element of complex type. The complex type definition is assigned the fully qualified name of the referenced record</td>
</tr>
<tr>
<td>Record Reference List</td>
<td>An element of complex type. The complex type definition is assigned the fully qualified name of the referenced record. When a global element declaration for a Record List parameter is referenced from a <code>&lt;sequence&gt;</code>, the maxOccurs attribute is set to &quot;unbounded&quot;.</td>
</tr>
<tr>
<td>Object</td>
<td>An element of anyType.</td>
</tr>
<tr>
<td>Object List</td>
<td>An element of anyType. When a global element declaration for an Object List is referenced from a <code>&lt;sequence&gt;</code>, the maxOccurs attribute is set to &quot;unbounded&quot;.</td>
</tr>
</tbody>
</table>

**Important!** If the input or output signature of a service is described by a record that contains unresolved links, when the WSDL generator attempts to create a WSDL document, a null pointer exception appears in the console. The WSDL generator creates an incomplete WSDL document and incomplete XML Schema definitions. An example of an unresolved link is a Record Reference variable where the referenced record cannot be found. Another example is a String, String List, or String Table variable with Content type property that cannot be found (the SAP BC schema containing the type definition has been moved or deleted).
If you select a custom SOAP processor as the directive, make sure the service for which you are generating the WSDL document conforms to the target service requirements for the custom processor. The following section explains the service requirements if you want to use the default SOAP processor to receive and send SOAP messages for the service. For information about creating custom SOAP processors, see the SAP BC SOAP Programming Guide.

**Service Requirements for Using the Default SOAP Processor**

If you want to use the default processor to handle the SOAP message request and response for the service, the service for which you are generating the WSDL document must meet the following requirements:

- The service must accept a soapRequestData object and a soapResponseData object as input and produce a soapResponseData object as output.
- The service must use the SOAP data-retrieval services, such as pub.soap.utils:getBody, pub.soap.utils:getHeader, and pub.soap.utils:getTrailers, to extract elements from SOAP message objects.
- The service might invoke other services to perform work on the extracted data.
- The service might use the SOAP message-composition services to populate soapResponseData or build a SOAP message string.

For more information about creating a target service for the default SOAP processor, see the SAP BC SOAP Programming Guide.

**Selecting an Input and Output Signature for a Service**

When you specify SOAP messaging as the protocol for accessing a service, you need to specify a record or XML Schema component to represent the input and/or output signature of the service. The signature describes the XML documents that the service expects as input and produces as output.

The WSDL generator does not create the input and output messages for the WSDL document from the signature specified on the **Input/Output** tab of the service. This is because target services for the default processor and custom SOAP processors must have a soapRequestData object and a soapResponseData as input and produce a soapResponseData object as output. While this signature requirement is necessary for the processors, it does not provide meaningful signature information for WSDL documents generated for the services used with these processors. In an XML Schema definition, a soapRequestData object and a soapResponseData object would be represented as elements of type **anytype**. To produce a meaningful, descriptive signature for the WSDL document, you can select a record, an element declaration, a complex type definition, or a simple type definition to represent the service input and output signature.
Keep the following items in mind when selecting a record or XML Schema component to describe the input or output signature:

- If you do not specify a record or XML Schema component for the input, the WSDL generator inserts an empty `<message>` element to represent the input in the WSDL document.

**Important!** If you do not specify a record or XML schema component to describe the service input and you indicate that the default SOAP processor will handle requests for this service, the WSDL document generated for this service will not be usable by Web Service consumers. When using the default SOAP message processor, you must select a record or XML Schema component to describe the input. For more information, see “Service Requirements for Using the Default SOAP Processor” on page 353.

- If you do not specify a record or XML Schema component for the output, the WSDL generator inserts an empty output `<message>` element into the WSDL document.

- If you specified a complex or simple type definition as the input or output signature, the `<part>` element for the input or output `<message>` element uses the type definition name as the value of the `type` attribute.

- If you specified an element declaration as the input or output signature, the `<part>` element for the input or output `<message>` element carries the element attribute and the WSDL generator uses the element name as the value of the `element` attribute.

- If you are using the default SOAP processor, the record or XML Schema component you select needs to contain at least one part (variable, element or type definition), and the name of the first part must correspond to the local name component of the service's universal name. If the service is assigned an explicit universal name, the first part name must match the local name for the explicit universal name. Otherwise, it must match the local name component of the implicit universal name.

This requirement is a result of the default SOAP processors routing behavior—the default processor routes messages to services by matching the fully expanded QName of the message body's first element to an explicit or implicit universal name. To generate a usable WSDL document, the input signature must use a naming convention that will work with the default processor. For example, if you select a record as the input signature, the name of the first variable in the record must match the local name component of the service's universal name.

**Note:** The WSDL generator does not support records that contain identically named variables at the top level or records with recursive references (a record that references itself). If you generate a WSDL document from a service that uses these unsupported record structures, the resulting XML Schema definition or WSDL document may be invalid.
The following procedure explains how to generate a WSDL document that specifies SOAP messaging as the protocol.

To generate a WSDL document that uses the SOAP Message protocol

1. In the Service Browser, select the service for which you want to generate a WSDL document.

2. On the Compose menu, select Generate WSD. Developer opens the Generate WSD dialog box.

```
Generate WSD dialog box
```

Developer populates the Namespace name and Local name fields only if you assigned an explicit universal name to the selected service. For information about assigning universal names to services, see the SAP BC SOAP Programming Guide.

3. In the Host field, type the numeric IP address or the domain name of the SAP BC Server on which the service will reside. You do not need to specify http:// or https:// as part of the host name. The WSDL generator automatically adds http:// or https:// when it compiles the network address for the service.

In the Host field, Developer automatically inserts the name of the server on which the service currently resides.

**Note:** If you are generating a WSDL document for a service located on your development server and you intend to distribute the document to partners, make sure that you enter the domain name or IP address of the production server in the Host field.
4 In the Port field, type the number of a port on the host server that you want to accept requests for this service. The port needs to accept HTTP or HTTPS requests.

In the Port field, Developer automatically inserts the port number you used to open the current session on SAP BC Server.

5 Under Protocol, select SOAP-MSG.

6 Under Via Transport, do one of the following:

<table>
<thead>
<tr>
<th>Select...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>Specify that requests to invoke the service must be sent via HTTP.</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Specify that requests to invoke the service must be sent securely using HTTPS.</td>
</tr>
</tbody>
</table>

Note: The transport you select must be the same as the type of requests that the port you selected accepts. If the port accepts HTTP requests, select HTTP under Transport. If the port accepts HTTPS requests, select HTTPS.

7 In the Directive list, select the process directive for the SOAP processor you want to use receive, process, and send SOAP messages that invoke the service. Select default if you want to use the default processor.

Note: The Directive list displays all of the registered soap processors on the server to which you are currently connected. If you plan to move the service to a production server, make sure the directive you select corresponds to a soap processor that is registered on the production server as well.

8 Next to the Input field, click to select a record, element declaration, or type definition to represent the input signature of the service. Developer opens the Select Input/Output Constraint dialog box.
Do one of the following to describe the input signature for the WSDL document.

— If you want to describe the input signature using a record, do the following:
  
  1 Under **Choose Constraint Type**, select **Record**.
  
  2 In the **Name** field, type the fully qualified name of the record that you want to use to describe the input signature.

  — or —

  Next to the **Folder** field, navigate to and select the record.

  3 Click **OK**.

— If you want to describe the input signature using an element declaration or a type definition from an XML Schema, do the following:

  1 Under **Choose Constraint Type**, select **Schema Component**.
  
  2 In the text field, after `http://`, type the Web location and name of the XML Schema that contains the element declaration or type definition that you want to use to describe the input signature.

**Note**: The XML Schema definition you enter must be located on the Web and must be accessible to Web Service consumers of the WSDL.
CHAPTER 14  Working with WSDL

3 Click **Load**. Developer groups the element declarations and type definitions under the headings ELEMENTS, SIMPLE TYPES, and COMPLEX TYPES.

4 Expand the headings to view the global element declarations or global type definitions in the XML Schema.

**Note:** If an XML Schema definition does not contain a component, the corresponding heading does not appear. For example, if the XML Schema does not contain simple type definitions, the SIMPLE TYPES heading does not appear.

5 Select the global element declaration, complex type definition, or simple type definition that you want to use to represent the input signature.

6 Click **OK**. Developer inserts the name of the selected element or type in the **Input** field.

10 Repeat steps 8 to 9 for the **Output** field.

11 In the **Target Namespace** field, type the URI you want to use as the target namespace for the WSDL document. In the generated WSDL document, the elements, attributes, and type definitions will belong to this namespace. By default, Developer displays the following as the target namespace:

http://host

Where *host* is the name of the SAP BC Server to which you are currently connected.

**Note:** Even if you delete the target namespace that Developer automatically enters, the WSDL generator will use http://host as the target namespaces in the WSDL document. If you want a null target namespace, you will need to edit the generated WSDL document.

12 Click **OK**. Developer displays the **Generate WSD** dialog box.

13 Under **Select Directory for Saving WSD** files, enter the directory to which you want to save the generated WSDL document and other generated files. Either select or enter an existing directory or type the path for a new directory. If you type the path for a new directory, the WSDL generator creates the directory when it generates the files.

14 Click **OK**.

When the WSDL generator finishes generating the WSDL document and the XML Schema definitions, Developer displays a message listing the files that were generated and the directory in which the files were placed.

If an error occurs during WSDL generation, Developer displays a message stating that an error occurred. If the WSDL generator created files before the error occurred, Developer displays a message listing the files that were generated and the directory in which the files were placed.
Generating a WSDL Document for a Service

For information about the generated files and how these files correspond to the service for which you generated the WSDL, see “What Files are Generated?” on page 344.

Note: If files of the same name exist already in the selected directory, SAP BC Developer displays a warning stating that files will be overwritten.

Generating a WSDL that Uses HTTP POST or GET

When you generate a WSDL document for a service, you can specify HTTP POST or HTTP GET as the protocol for invoking the service. If you specify HTTP-POST as the protocol, you can select text/xml or URL encoding as the input format for the service. When you select HTTP GET as the protocol, Developer automatically sets the input format to URL encoding.

When you specify HTTP POST or HTTP GET as the protocol for the WSDL document, keep the following points in mind:

- When you select URL encoding as the input format, the WSDL generator uses the input parameters declared on the **Input/Output** tab of the service to create the input message for the WSDL document. However, for URL encoding, the input signature can only contain String and String List variables. The input signature should not contain Record, Record List, Objects, Object Lists, or String Table variables because these variables cannot be represented in name=value pairs in the HTTP request.

- When you select text/xml as the input format, you can select a record or XML Schema component (element declaration, complex type definition, or simple type definition) to describe the incoming XML document. For more information about using records or XML Schema components to describe the service input, see “Selecting the Input and Output Signature for HTTP GET and POST” on page 360.

- For the HTTP protocols, Developer selects text/xml as the output format. You need to select a record or XML Schema component (element declaration, complex type definition, or simple type definition) to represent the outbound XML document.

For information about describing the service input and output for the HTTP protocols, see ”Selecting the Input and Output Signature for HTTP GET and POST” on page 360.
Service Requirements for Using HTTP POST or HTTP GET as the Protocol

If you want the WSDL document to specify HTTP POST or GET as the protocol for invoking the service, the service for which you are generating the WSDL document must meet the following requirements:

- If you select HTTP POST as the protocol and you select text/xml as the input format, the service input signature must include a node object as an input parameter. When SAP BC Server receives an HTTP POST request where the Content-Type header is text/xml, the server automatically parses the body of the request (the XML document) and passes it as a node to the service specified in the request’s URL.

Because a node object variable does not provide meaningful signature information for the WSDL document, you can select a record or XML Schema component to describe the input signature. The record or XML Schema component you select describes the XML document the service expects as input.

- The service must return an XML document. When generating a WSDL document, the WSDL generator assumes that all services invoked via an HTTP request will return an XML document (text/xml content). To return an XML document, a service can do one of the following:
  - Invoke the pub.flow:setResponse service. This service takes any string and returns it as the body of an HTTP response. For more information about the pub.flow:setResponse service, see the SAP BC Built-In Services Guide.
  - Create and assign an XML output template to the service. You can use an XML output template to extract values from the pipeline and insert those values as element content in an XML tag. For more information about output templates, see Building Output Templates and DSPs.

Note: If you do not use the pub.flow:setResponse service or an XML output template to return an XML document, then an HTML document will be returned to the HTTP request. However, most Web Service consumers expect a service described by WSDL to return an XML document.

Selecting the Input and Output Signature for HTTP GET and POST

When you specify HTTP POST or GET as the protocol for accessing a service, you may need to specify a record or XML Schema component to represent the input and/or output signature of the service. The signature describes the XML documents the service expects as input and produces as output.
You need to specify a record or XML Schema component for the signature if:

- You selected HTTP POST as the protocol and specified text/xml as the input format.
- You selected HTTP POST or HTTP GET as the protocol and the service returns output to the requesting client. (If the service describes a one-way WSDL operation, you do not need to specify a record or XML Schema component for output.)

By selecting a record or an XML Schema component (element declaration, simple type definition, or complex type definition), the WSDL generator can produce a meaningful, descriptive signature for the WSDL document. Keep the following items in mind when selecting a record or XML Schema component to describe the service input or output:

- If you do not specify a record or XML Schema component for the input, the WSDL generator creates an empty <message> element to represent the input in the WSDL document.
- If you do not specify a record or XML Schema component for the output, the WSDL generator creates a WSDL document describing a one-way operation. Because a one-way operation does not expect a response, the WSDL generator does not create an output <message> element.
- If you specified a complex or simple type definition as the input or output signature, the <part> element for the input or output <message> element carries the type attribute and the WSDL generator uses the type definition name as the value of the type attribute.
- If you specified an element declaration as the input or output signature, the <part> element for the input or output <message> element carries the element attribute and the WSDL generator uses the element name as the value of the element attribute.

**Note:** The WSDL generator does not support records that contain identically named variables at the top level or records with recursive references (a record that references itself). If you generate a WSDL document from a service that uses these unsupported record structures, the resulting XML Schema definition or WSDL document may be invalid.

The following procedure explains how to create a WSDL document that specifies HTTP POST or HTTP GET as the protocol.

**To generate a WSDL document that specifies HTTP POST or HTTP GET as the protocol**

1. In the Service Browser, select the service for which you want to generate a WSDL document.
2. On the Compose menu, select Generate WSD. Developer opens the Generate WSD dialog box.
3 In the **Host** field, type the numeric IP address or the domain name of the SAP BC Server on which the service will reside. You do not need to specify http:// or https:// as part of the host name. The WSDL generator automatically adds http://or https:// when it compiles the network address for the service.

   In the **Host** field Developer automatically inserts the name of the server on which the service currently resides.

   **Note:** If you are generating a WSDL document for a service located on your development server and you intend to distribute the document to partners, make sure that you enter the domain name or IP address of the production server in the **Host** field.

4 In the **Port** field, type the number of a port on the host server that you want to accept requests for this service. The port needs to accept HTTP or HTTPS requests.

   Developer automatically inserts the port number you used to open the current session on SAP BC Server.

5 Under **Protocol**, select one of the following:

   **Select...** | **To...**
   --- | ---
   HTTP-POST | Specify HTTP POST as the protocol to communicate with the service.
   HTTP-GET | Specify HTTP GET as the protocol to communicate with the service.
Generating a WSDL Document for a Service

6 Under **Via Transport**, do one of the following:

<table>
<thead>
<tr>
<th>Select...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>Specify that requests to invoke the service must be sent via HTTP.</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Specify that requests to invoke the service must be sent securely</td>
</tr>
<tr>
<td></td>
<td>using HTTPS.</td>
</tr>
</tbody>
</table>

**Note:** The transport you select must be the same as the type of requests that the port you selected accepts. If the port accepts HTTP requests, select **HTTP** under **Transport**. If the port accepts HTTPS requests, select **HTTPS**.

7 In the **Directive** field, specify the invoke directive used to invoke a service on SAP BC Server. Developer automatically displays “invoke” as the directive.

**Important!** For most server configurations, this value should not be changed.

8 In the **Path** field, specify the path name of the service in the `folder.subFolder/serviceName` format. Separate subfolders with a “.” (period). Separate the service name from the folder name with a “/” (forward slash). Developer automatically enters the path name of the service.

**Note:** If you plan to move the service to a different folder after generating the WSDL document, make sure the path you specify represents the final location of the service at production time.

9 If you selected **HTTP-GET** in step 5, skip to step 12.

10 If you selected **HTTP-POST** in step 5, in the **Input format** list, select the format in which the service expects input.

<table>
<thead>
<tr>
<th>Select...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>text/xml</td>
<td>Specify that input to the service needs to be supplied in an XML</td>
</tr>
<tr>
<td></td>
<td>document.</td>
</tr>
<tr>
<td></td>
<td>If you select <strong>text/xml</strong>, you need to select a record or XML Schema</td>
</tr>
<tr>
<td></td>
<td>component to describe the input signature of the service.</td>
</tr>
<tr>
<td>URL encoded</td>
<td>Specify that input to the service needs to be submitted in</td>
</tr>
<tr>
<td></td>
<td>name=value pairs in the URL that requests the service.</td>
</tr>
<tr>
<td></td>
<td>If you select <strong>URL encoded</strong>, the WSDL generator uses the input</td>
</tr>
<tr>
<td></td>
<td>signature on the <strong>Input/Output</strong> tab of the service to create the</td>
</tr>
<tr>
<td></td>
<td>input message for the WSDL document.</td>
</tr>
</tbody>
</table>
11 If you selected text/xml as the input format in step 9, do one of the following to describe the input signature for the WSDL document.

   — If you want to describe the input signature using a record, do the following:

      1 Next to the Input field, click . Developer opens the Select Input/Output Constraint dialog box.
      2 Under Choose Constraint Type, select Record.
      3 In the Name field, type the fully qualified name of the record that you want to use to describe the input signature.
         —or—
         Next to the Folder field, navigate to and select the record.
      4 Click OK.

   — If you want to describe the input signature using an element declaration or a type definition from an XML Schema, do the following:

      1 Next to the Input field, click . Developer opens the Select Input/Output Constraint dialog box.
      2 Under Choose Constraint Type, select Schema Component.
      3 In the text field, after http://, type the Web location and name of the XML Schema that contains the element declaration or type definition that you want to use to describe the input signature.
      4 Click Load. Developer groups the element declarations and type definitions under the headings ELEMENTS, SIMPLE TYPES, and COMPLEX TYPES. Expand the headings to view the global element declarations or global type definitions in the XML Schema.

Note: If an XML Schema definition does not contain a component, the corresponding heading does not appear. For example, if the XML Schema does not contain simple type definitions, the SIMPLE TYPES heading does not appear.

   5 Select the global element declaration, complex type definition, or simple type definition that you want to use to represent the input signature.
   6 Click OK. Developer inserts the name of the selected element or type in the Input field.

12 To describe the XML document the service produces as output, follow step 11 for the Output format field.
13 In the **Target Namespace** field, type the URI you want to use as the target namespace for the WSDL document. In the generated WSDL document, the elements, attributes, and type definitions will belong to this namespace. By default, Developer displays the following as the target namespace:

http://host

Where *host* is the name of the SAP BC Server to which you are currently connected.

**Note:** Even if you delete the target namespace that Developer automatically enters, the WSDL generator will use http://host as the target namespaces in the WSDL document. If you want a null target namespace, you will need to edit the generated WSDL document.

14 Click **OK**. Developer displays the **Generate WSD** dialog box.

15 Under **Select Directory for Saving WSD** files, enter the directory to which you want to save the generated WSDL document and other generated files. Either select or enter an existing directory or type the path for a new directory. If you type the path for a new directory, the WSDL generator creates the directory when it generates the files.

16 Click **OK**.

When SAP BC Server finishes generating the WSDL document and the XML Schema definitions, Developer displays a message listing the files that were generated and the directory in which the files were placed.

If an error occurs during WSDL generation, Developer displays a message stating that an error occurred. If the WSDL generator created files before the error occurred, Developer displays a message listing the files that were generated and the directory in which the files were placed.

For information about the generated files and how these files correspond to the service for which you generated the WSDL, see “What Files are Generated?” on page 344.

**Note:** If files of the same name exist already in the selected directory, SAP BC Developer displays a warning stating that files will be overwritten.
Creating a Web Service Connector

A Web Service Connector is a flow service that invokes a Web Service located on a remote server. Developer uses a WSDL document to automatically generate a Web Service Connector. Developer assigns the Web Service Connector an input and output signature that corresponds to the input and output messages from the WSDL document.

Tip! Because a Web Service Connector is a flow service, you can invoke, test, debug, and lock a Web Service Connector the same way you would a flow service.

The following procedure explains how to create a Web Service Connector from a WSDL document.

To create a Web Service Connector

2. In the New dialog box, select Web Service Connector, and click Next.
3. In the Folder tree, select the folder in which you want to save the Web Service Connector and its supporting SAP BC elements, and click Next.
4. In the New Web Service Connector dialog box, under Choose a .wsd or .wsdl file by entering the URL or selecting a local file, do one of the following:
   - If you want to create a Web Service Connector from a .wsd or .wsdl file that resides on the Internet, enter the URL of the file. (The URL must begin with http:// or https://.)
   - If you want to create a Web Service Connector from a .wsd or .wsdl file that resides on your local file system, click to navigate to and select the file.
5. Click Finish.

Developer creates the Web Service Connector and its supporting SAP BC elements and saves everything in the folder you specified. For information about the SAP BC elements that Developer generates, see “What Elements Are Generated?” on page 367.

While creating the Web Service Connector, Developer validates the WSDL document. If Developer cannot create a Web Service Connector from the WSDL document or cannot completely generate the Web Service Connector because of an invalid WSDL document or missing WSDL elements, Developer displays error messages or warning messages. For more information about the error and warning messages that occur during Web Service Connector generation, see “WSDL Errors and Warnings” on page 647.
**What Elements Are Generated?**

When Developer creates a Web Service Connector from a WSDL document that you select, it also creates some supporting SAP BC elements. Each of the SAP BC elements that Developer creates corresponds to a WSDL element in the WSDL document.

The following table identifies the SAP BC elements that Developer may create when it generates a Web Service Connector.

<table>
<thead>
<tr>
<th>Developer creates a...</th>
<th>That corresponds to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Folder</td>
<td>Each unique <code>&lt;portType&gt;</code> element in the WSDL document. The subfolder name corresponds to the <code>portTypeName</code>.</td>
</tr>
<tr>
<td>Web Service Connector</td>
<td>Each unique <code>&lt;operation&gt;</code> element in the <code>&lt;portType&gt;</code> element. The Web Service Connector name corresponds to the operation name.</td>
</tr>
<tr>
<td>rec folder</td>
<td>All of the records generated from the messages in the WSDL document.</td>
</tr>
<tr>
<td>Record</td>
<td>Each <code>&lt;message&gt;</code> element in the WSDL document. The record name corresponds to the message name.</td>
</tr>
<tr>
<td>SAP BC schema</td>
<td>Each namespace to which the element declarations, attribute declarations, and type definitions that define the message parts (input and output signature) belong. The SAP BC schema name follows the naming convention <code>schema_messageName</code>.</td>
</tr>
</tbody>
</table>

**Note:** After Developer generates the Web Service Connector, you may need to edit the Web Service Connector. For example, you might need to set user name and password values for invoking the Web Service on the remote server.

**Note:** When creating the record for the input message, Developer inserts the `_port` variable into the record. The Web Service Connector uses the `_port` variable as the switch value in the BRANCH `on '/_port'` step to determine which network address and binding to use to invoke the Web Service.
Example Web Service Connector

The following Web Service Connector was generated from a WSDL document that describes a Web Service that authorizes a credit card. The WSDL document specified:

- An input message named AuthorizeCreditCardInput that specifies the inputs Name, CreditCardType, CreditCardNumber, and ExpirationDate.
- An output message named AuthorizeCreditCardOutput that specified the output isAuthorized.
- A <binding> element that specified SOAP RPC as the protocol.
- A <service> element that contained one <port> named AuthorizeCreditCardPortType.
- A single <operation> element named AuthorizeCreditCard.

On the Input/Output tab for the Web Service Connector, notice that the Web Service Connector uses references to the input and output records to define the service signature.
Developer inserts flow steps into the Web Service Connector by following an internal template for inserting input data into the service request, sending the request, processing the response, and adding service output values to the pipeline. The template that Developer follows depends on the protocol specified in the WSDL document. The following illustration shows the Web Service Connector generated for the Web Service that performs credit card authorization.
CHAPTER 14 Working with WSDL

Flow Editor for a Web Service Connector

This BRANCH step contains a child step for each named port.

This BRANCH step contains a child step for each named binding.

This SEQUENCE corresponds to a binding for the SOAP RPC protocol.

Note: The $default step handles cases where the value of the _port variable at run time does not specify a valid port name. The $default port corresponds to the first named <port> element in the WSDL document.
Passing XML Data to a Service

- Sending XML Documents to SAP BC Server ........................................ 372
- Submitting an XML Document in a String Variable .......................... 372
- Submitting an XML Document in $xmlData ..................................... 373
- Posting an XML Document via HTTP ............................................. 375
- Submitting an XML Document via FTP .......................................... 376
- Submitting an XML Document via Email ......................................... 378
## Sending XML Documents to SAP BC Server

With the load and query services, SAP BC Server provides ways for you to fetch XML documents from the Web. However, the server also provides the following automated mechanisms for receiving arbitrary XML documents, parsing them, and passing them as input to a specified service.

- A client can submit an XML document to a service in a string variable (of any name), which the target service can convert into a node object using `web.pub:stringToDocument`.
- A client can submit an XML document to a service in a special string variable named `$xmlData`, and SAP BC Server will automatically parse the variable and pass it to the service as a node object.
- A client can post an XML document to a service via HTTP.
- A client can FTP an XML document to a service.
- A client can Email an XML document to a service.
- A client can send the XML document as an e-mail attachment.

### Submitting an XML Document in a String Variable

One way to submit an XML document to SAP BC Server is to pass the entire document to a service in a String variable. When you use this approach, you should code the target service to execute `pub.web:stringToDocument` to convert the String variable (i.e., the XML document) to a node object. Once the document is represented as a node object, it exists in a form that can be queried or converted to an IData object. For information about using `stringToDocument`, see the SAP BC Built-In Services Guide.

The following code fragment shows how a Java client might submit an XML document to a service called `purch:postOrder` on SAP BC Server. In this example, the client 1) loads the XML document into a String, 2) puts the String into the element orders in an IData object named `inputs`, and 3) invokes `purch:postOrder` on the server at localhost:5555.

```java
// Load the XML document into a String
String xmlDocument = loadXML();

// Put the String into the element orders in an IData object
IData inputs = ...;
inputs.setValue(Element.orders, xmlDocument);

// Invoke the purch:postOrder service on the server
purch:postOrder(inputs);`
import com.wm.app.b2b.client.*;
import com.wm.util.*;
import com.wm.data.*;
import java.io.*;
public class ArbitraryXMLClient
{
    /*--Load XML into orders string*/
    String orders = YourLoadXMLMethod(orderFile);

    /*--Put input values into IData object*/
    IData inputs = IDataFactory.create();
    IDataCursor inputsCursor = inputs.getcursor();
    inputsCursor.last();
    inputsCursor.insertAfter("orders", orders);
    inputsCursor.insertAfter("authCode", authCode);

    /*--Submit request to server*/
    c.connect("localhost:5555", "null", null);
    IData outputs = c.invoke("purch", "postOrder", inputs);
    c.disconnect();
}

On the server, purch:postOrder should be coded to pass the XML document in orders to pub.web:stringToDocument. This will produce a node object that can be subsequently queried or converted to an IData object. For information about the services that you can use to manipulate node objects, see pub.web:queryDocument and pub.web:documentToRecord in the SAP BC Built-In Services Guide.

Submitting an XML Document in $xmldata

Submitting an XML document to SAP BC Server using $xmldata is similar to submitting it as a String variable, but in this case, the service receiving the document does not need to include a parsing step—SAP BC Server automatically parses the contents of $xmldata.

The name $xmldata has special meaning to SAP BC Server—it assumes the value of this variable is an XML document. When it receives a request that includes an input variable named $xmldata, the server automatically parses the contents of that variable, producing a document object that can be used by any service that takes a node as input.

The following example shows a client application that reads an XML document from a file, assigns the text in the file to $xmldata and then passes $xmldata to a service called sales:getOrder.
import com.wm.app.b2b.client."
import com.wm.util."
import com.wm.data."
import java.io."

public class ArbitraryXMLClient
{
    public static void main(String args[])
    throws Exception
    {
        //--Read XML document from a specified file (or from stdin)
        Context c = new Context();
        IData inputs = IDataFactory.create();
        IDataCursor inputsCursor = inputs.getCursor();
        Reader in = null;
        if(args.length > 0)
        {
            in = new InputStreamReader(new FileInputStream(args[0]));
        } else
        {
            in = new InputStreamReader(System.in);
        }
        char[] buf = new char[8192];
        int count = 0;
        StringBuffer sb = new StringBuffer();
        while((count = in.read(buf)) != -1)
        {
            sb.append(buf, 0, count);
        }
        //--Assign XML document to string variable
        String xmldata = sb.toString();
        //--Put XML document into $xmldata in IData object
        inputsCursor.last();
        inputsCursor.insertAfter("$xmldata", xmldata);
        //--Submit request to server
        c.connect("localhost:5555", "null", null);
        IData outputs = c.invoke("sales", "getOrder", inputs);
        c.disconnect();
        //--Display the returned output values
        System.out.println(outputs);
    }

The service invoked by this client must be a service that takes a node as an input variable (e.g., queryDocument, documentToRecord), since this is what SAP BC Server produces when it receives this request.
Posting an XML Document via HTTP

**Important!** The example above shows a Java-based client; however, any type of SAP BC client can be used, even a browser-based client. With a browser-based client, you would post the XML document as the value portion of a $xmlData=value pair. You would most likely construct the XML document with Javascript. You may post other name=value pairs with the request.

### Posting an XML Document via HTTP

A client can post an XML document to a service via HTTP. To use this approach, you must have a client that can:

- Send a string of data (an XML document) to SAP BC Server using the HTTP POST method.
- AND–
- Set the value of the Content-Type request-header field to text/xml.

When SAP BC Server receives an HTTP POST request where Content-Type is text/xml, it automatically parses the body of the request (the XML document) and passes it as a node to the service specified in the request's URL.

Since most browsers do not allow you to modify the Content-Type header field, they are not suitable clients for this type of submission. Clients that you might use to submit an XML document in this manner are: PERL scripts (which allows you to build and issue HTTP requests) or the SAP BC Server service, pub.client:http.

Regardless of which client you use, it must do the following:

- Submit a POST request to SAP BC Server
- Address the request to the URL of an service (e.g., http://rubicon:5555/invoke/purch/postOrder)
- Set the Content-Type header field to text/xml
- Contain an XML document in the body of the message. The document must be the only text that appears in the body of the request. Do not assign it to a name=value pair.

**Important!** When you submit the XML document, place an extra carriage return/new line (\r\n) at the end of it to indicate the end of the document.
The following example describes the values that you set if you use pub.client:http to POST an XML document to a service.

<table>
<thead>
<tr>
<th>Set this Variable...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the service that you want to invoke. The following value would invoke the purch.postOrder service from a server named “rubicon” with port number “5555.”</td>
</tr>
<tr>
<td>method</td>
<td>post</td>
</tr>
<tr>
<td>headers.Content-Type</td>
<td>text/xml</td>
</tr>
<tr>
<td>data.string</td>
<td>The XML document that you want to post.</td>
</tr>
<tr>
<td>data.bytes</td>
<td>-OR-</td>
</tr>
</tbody>
</table>

You will also set any optional HTTP parameters, such as authorization information, that are required by your application.

**Submitting an XML Document via FTP**

You can FTP an XML document to SAP BC Server’s FTP listening port. (By default the FTP port is assigned to port 8021. However, this assignment is configurable, so you should check with your SAP BC Server administrator to see which port is used for FTP communications on your SAP BC Server.)

When SAP BC Server receives an XML document on the FTP listening port, it automatically parses the document and passes it as a node to the service in the directory where the file was FTPed.

To submit an XML document to SAP BC Server via FTP:

- The XML document must be contained in a file that has a file extension of “xml.”
- The service to which you want to pass the document must take a node as input.
Building a Client that Sends an XML Document to a Service

If you want to submit an XML document to a service through FTP, the application sending the document must do the following:

1. Initiate an FTP session on SAP BC Server's FTP listening port.
2. Point to the directory that contains the service to which you want to pass the XML document.

   Example  cd \ns\Purchasing\SubmitOrder

**Important!** Note that the root directory for this operation is your SAP BC Server's namespace directory (ns), not the root directory of the target machine. Therefore, if you want to FTP a file to a service in the Purchasing package, you use \ns\Purchasing\ServiceName as the path to that service, not \<sapbc>\server\ns\Purchasing\ServiceName.

3. Copy the XML document to this directory using the following command:

   put XMLDoc.xml

   Where XMLDoc.xml is the name of the file that you want to pass to SAP BC Server.

   Example  put PurchaseOrder.xml

   Note that the document you FTP to SAP BC Server is never actually written to the server’s file system. The document you send and the output file it produces (see below), are written to a virtual directory system maintained in your SAP BC session.

FTPing a File From a SAP BC Server

The pub.client folder contains built-in services you can use to FTP a file from a SAP BC Server. For information about these services, see the *SAP BC Built-In Services Guide*.

Getting Results from an FTPed Document

The results from a service executed by an FTP request are written to the same virtual directory where the XML document was initially FTPed.

- If the service does not have an output template assigned to it, the results (i.e., the contents of the pipeline) are XML-encoded and returned as an XML document.
- If the service has an XML output template assigned to it, that template is applied to the results. (If the template is not an XML-based template, it is not applied).
The name of the output file to which results are written is:

```
XMLDoc.xml.out
```

Where `XMLDoc.xml` is the name of the XML file initially FTPed to the service. You retrieve this document using the FTP “get” command. For example, if you put a document called `PurchaseOrder.xml` on SAP BC Server, you would use the following FTP command to get its results:

```
Example get PurchaseOrder.xml.out
```

When you end the FTP session, SAP BC Server automatically deletes the original file and its results from your session.

**Submitting an XML Document via Email**

You can email an XML document to an email mailbox and have SAP BC Server automatically retrieve the email message and process the XML document it contains. To do this, your SAP BC Server must be configured with an email port that monitors the mailbox to which the XML document will be sent. (Consult your SAP BC Server administrator to see whether an email port has been set up on your SAP BC Server.)

When an XML document arrives in the email port's mailbox, SAP BC Server automatically retrieves the message, parses the attached XML document, and passes that document as a node to the service specified on the email's subject line (or, if a service is not specified on the subject line, the email port's default service).

**Requirements for Sending an XML Document via Email**

To submit an XML document to SAP BC Server via email, your client program must:

- Put the XML document in an email attachment
- Set the email's `Content-Type` header to `text/xml`
- Specify the name of the service that will process the document in the email's subject line. If you leave the subject line empty, the document will be processed by the global service if one is defined or, if not, by the default service assigned to the email port (if one has been assigned). For information about specifying the port's default service, see the [SAP BC Administration Guide](#).

The service that will process the XML document must take a node as input.
The following example describes the values that you would set if you used pub.client:smtp to email an XML document to a service. (For more information about using this service, see the SAP BC Built-In Services Guide.)

<table>
<thead>
<tr>
<th>Set this Variable...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>to</td>
<td>A String specifying the email address that SAP BC Server’s email port monitors.</td>
</tr>
<tr>
<td>subject</td>
<td>A String specifying the fully qualified name of the service that SAP BC Server will pass the attached document to.</td>
</tr>
<tr>
<td>Example: orders:ProcessPO</td>
<td></td>
</tr>
<tr>
<td>from</td>
<td>A String containing the email address to which the results of the service will be sent.</td>
</tr>
<tr>
<td>body</td>
<td>A String containing input parameters for the service in URL query string format. The following example sets five parameters: one, two, and three are set to the values 1, 2 and 3, respectively. The parameters $user and $pass have special meaning to the email port. You use these parameters to specify the user name and password for the email port. You must specify these two parameters if authentication is enabled on the email port.</td>
</tr>
<tr>
<td>Example: one=1&amp;two=2&amp;three=3&amp;$user=Administrator&amp;$pass=manage</td>
<td></td>
</tr>
<tr>
<td>attachments.contenttype</td>
<td>A String set to: text/xml</td>
</tr>
<tr>
<td>attachments.content</td>
<td>The byte [] or a String containing the XML document.</td>
</tr>
<tr>
<td>–OR–</td>
<td></td>
</tr>
<tr>
<td>attachments.filename</td>
<td>A String specifying the fully qualified name of the file containing the XML document.</td>
</tr>
</tbody>
</table>

### Getting Results from an Emailed Document

If your email port has been configured to return results, then the results from a service invoked through the port are emailed back to the sender of the original message, in an attachment file called xml.out.

- If the service does not have an output template assigned to it, the results from the service (i.e., the contents of the pipeline) are XML-encoded and returned as an XML document.
If the service has an output template assigned to it, that template is applied to the results.

**Important!** By default, the email port does not return any results from requests that it receives. If you want the port to return results, you must explicitly configure it to do so. For information about configuring the email port to return results, see your SAP BC Administration Guide.
Using the Load and Query Services

- What Are the Load and Query Services? ........................................ 382
- Basic Concepts ........................................................................ 382
- Using the loadDocument Service ............................................. 383
- Using the queryDocument Service .......................................... 400
- Load and Query Shortcuts ...................................................... 427
What Are the Load and Query Services?

SAP BC Server is equipped with a set of “load and query” services that allow you to fetch HTML or XML documents via HTTP or HTTPS and selectively extract information from them for other services. Through these services, you can connect a service to virtually any document on the Internet. (If you want to retrieve documents from a local file system, see pub.file:getFile in the SAP BC Built-In Services Guide.)

Fetching a document from the Internet with SAP BC Server is a two-step process.

- First, you use the pub.web:loadDocument service to retrieve the document that contains the information you need.
- Next, you extract the pieces of information you need to use and assign them to SAP BC variables. For this step, you can use the pub.web:queryDocument service (to select specific elements from the document) or the pub.web:documentToRecord service (to convert all elements in the document to variables).

Basic Concepts

To successfully use SAP BC Server’s load and query services, you should understand the following terms and concepts.

What Is Parsing?

Parsing is the operation that the server performs to convert an HTML or XML document (a string) into a Document object whose elements can be addressed and extracted by services. SAP BC Server automatically parses documents that you fetch with the loadDocument service.
What Is a Document Object?

A Document object is the result of a parsing operation. It is a node-based representation of an entire XML or HTML document. The Document object expresses a document in a tree-like structure that allows the data within it to be efficiently addressed and mapped into services.

Once an HTML or XML document is converted to a Document object, you can manipulate it (e.g., extract specific information from it or convert it into an IData object) using any service that takes a document or node variable as input.

What Is a Node?

A node is a Document object or an element of a Document object. Services that require a node as input or produce a node as output have the variable name node declared as an input or output parameter.

Important! You may use a Document object anywhere a node variable is specified; however, you cannot use a node when a document variable is specified. Because a node is a subset of a Document object, it does not include certain document-specific elements that are inherently part of a Document object.

What Is a Query?

With respect to XML and HTML documents, a query is an expression, written in the XML Query Language (XQL) or the webMethods Query Language (WQL) that you use to extract (filter) information from XML or HTML documents. (WQL was referred to as “WOM” in earlier releases of SAP BC Server.)

Using the loadDocument Service

You use the pub.web:loadDocument service to retrieve an XML or HTML document from the Internet. This service does the following:

- It submits a HTTP or HTTPS request for a specified XML or HTML document.
- AND THEN–
- It parses the returned document.

The output from loadDocument is a Document object that can be used with any service that takes a document or a node as input.

Note: If you want to fetch a document from a local file system, do not use loadDocument. Instead, use the pub.file:getFile service.
CHAPTER 16 Using the Load and Query Services

Adding loadDocument to a Flow Service

The following procedure describes the steps you use to insert pub.web:loadDocument into a flow service.

**Important!** The following procedure explains how to manually insert and specify the loadDocument service. If you are constructing a “load and query” service, however, you may want to use the wizard instead. This wizard allows you to build a load and query sequence by pointing to a URL or recording the actions in a browser. For information about using the wizard to create a load and query sequence, see “Load and Query Shortcuts” on page 427.

To insert loadDocument into a flow service

1. In the Service Browser, select the service in which you want to insert loadDocument.
2. Click on the Flow Editor toolbar, and select loadDocument. (If loadDocument does not appear on the menu, click Browse and select pub.web:loadDocument from the list next to Folder.)
3. Click the Pipeline tab if it is not already displayed.
4. Complete the following steps to specify the URL of the document that you want to retrieve.
   1. In Service In, select url.
   2. Click on the toolbar.
   3. Type the URL, starting with http: or https:, and then click OK.

   **Examples**
   The following string would retrieve the specified document via http:

   http://www.rubicon.com/orders/orders.xml

   The following string would retrieve the document via https:

   https://www.rubicon.com/orders/orders.xml

5. Use the Set Value modifier to assign values to the remaining input variables as needed. For more information about the input variables for loadDocument, see “Inputs for loadDocument” on page 385.

Input and Output Values for loadDocument

The inputs for loadDocument specify what document you want to retrieve, what data (if any) you want to submit with the request, and how you want SAP BC Server to parse the document it receives.
When you use loadDocument in a flow service, you use the Set Value modifier in the Pipeline Editor to assign values to these variables. If you call loadDocument from a program, your program must set these variables in theIData object it passes to the pub.web:loadDocument service.

### Service Inputs for loadDocument

<table>
<thead>
<tr>
<th>Pipeline In</th>
<th>Service In</th>
<th>Service Out</th>
<th>Pipeline Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline In</td>
<td>Service In</td>
<td>Service Out</td>
<td>Pipeline Out</td>
</tr>
<tr>
<td>Pipeline In</td>
<td>Service In</td>
<td>Service Out</td>
<td>Pipeline Out</td>
</tr>
<tr>
<td>Pipeline In</td>
<td>Service In</td>
<td>Service Out</td>
<td>Pipeline Out</td>
</tr>
<tr>
<td>Pipeline In</td>
<td>Service In</td>
<td>Service Out</td>
<td>Pipeline Out</td>
</tr>
</tbody>
</table>

Use the Set Value modifier to assign input values to loadDocument.

### Inputs for loadDocument

The following describes the input values used by the loadDocument service.

**The url variable**

Set this variable to specify the URL of the document you want to retrieve.

**Example**  

http://www.rubicon.com/orders/orders.html

- The value you specify must be a string that starts with the protocol specifier “http:” or “https:”

- You may include a query string (for example, a collection of “name=value” pairs) with the string that you specify. However, we suggest that you use the data variable for this type of information instead. It is usually a more practical place for “name=value” data, because it allows you to map individual variables in the query string. See “The data variable” on page 388.
CHAPTER 16 Using the Load and Query Services

To assign url in a flow service

Do one of the following:

- Use the Map modifier to assign a pipeline variable to url.
  -OR-

- Use the Set Value modifier in the Pipeline Editor to specify the URL you want to use.

The method variable

Set this value to specify the HTTP method (GET or POST) that you want the target server to execute on the resource specified in url. This value determines the way in which loadDocument submits data values (if any) to the resource identified in url.

- If you set method to “get”, loadDocument appends the values you specify in data to the value in url. (Note that only certain data elements are valid when you use the GET method).
- If you set method to “post”, loadDocument sends the values in data in the body of the HTTP or HTTPS request.

To assign method in a flow service

Do one of the following:

- Use the Map modifier to assign a pipeline variable to method.
  -OR-

- Use the Set Value modifier in the Pipeline Editor to select the method you want to use.

Setting the method variable in a flow service

Select get or post from the list.
The auth variable

You use this variable to specify the authorization and authentication credentials (e.g., user name and password) that you want loadDocument to submit when it requests the specified document.

To set the auth variable correctly, you must know:

- Whether the Internet resource you are accessing is protected, and if so, what type of authentication is required to access it. (If the resource is not protected—i.e., it does not require a user name and password—you do not need to set any of the elements in auth.)
- The user name and password that is required by the resource you are requesting.

If the resource you are requesting is protected, you must specify the auth variable when you use loadDocument. Each element in auth specifies a piece of the “Authorization” field that loadDocument will submit to the target server.

Set this element... To specify...

type

The protection scheme used to control access to the requested document. This element must have one of the following values.

<table>
<thead>
<tr>
<th>Set type to...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>Use the Basic Protection Scheme.</td>
</tr>
</tbody>
</table>

user

A user name that has authority to access the resource specified in url. This value defaults to the value of watt.net.httpUser in the server's configuration file (server.cnf).

pass

The password associated with the user name specified in user. If the user does not require a password, leave pass empty. This value defaults to the value of watt.net.httpPass in the server’s configuration file (server.cf).

To assign auth in a flow service

Do one of the following:

- Use the Map modifier to assign a pipeline variable to the type, user, and/or pass elements as needed.
- OR-
- Use the Set Value modifier in the Pipeline Editor to hard-code the type, user, and/or pass elements as needed.
CHAPTER 16  Using the Load and Query Services

The data variable

Use the data variable to specify data that you want loadDocument to submit to the resource identified in url.

**Important!** If you include your data with the string in url, do not specify a value in data.

The data variable is a Record that contains several predefined elements. Each element allows you to specify data in a different way. (Note that some elements are valid for only one HTTP method—GET or POST.)

**Note:** If you submit data using the args or table element, loadDocument automatically sets the HTTP Content-Type header to application/x-www-form-urlencoded. If you want to specify a different Content-Type, use the string or bytes element to submit your data, not args or table.

<table>
<thead>
<tr>
<th>Use this element...</th>
<th>If you want to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>args</td>
<td>Specify “name=value” pairs as a Record of string elements. When you specify data in this manner, loadDocument automatically appends the elements from args to url (if you are using the GET method) or inserts them in the body of the message (if you are using the POST method). You use this variable.</td>
</tr>
<tr>
<td></td>
<td>To specify data in this manner, create one string element for each “name=value” pair that you want to submit, where:</td>
</tr>
<tr>
<td></td>
<td>- The name of the element defines the name portion of the pair, and...</td>
</tr>
<tr>
<td></td>
<td>- The value of the element specifies the value portion of the pair.</td>
</tr>
<tr>
<td></td>
<td>Note that loadDocument will automatically do the following:</td>
</tr>
<tr>
<td></td>
<td>- URL-encode values you specify.</td>
</tr>
<tr>
<td></td>
<td>- Insert the “&amp;” character between each name=value pair that it constructs from the args (for POST and GET). You do not have to include this character in the values you specify.</td>
</tr>
<tr>
<td></td>
<td>- Insert the “?” character at the beginning of the query string it generates from args for the GET method. You do not have to include this character in the values you specify.</td>
</tr>
<tr>
<td></td>
<td><strong>Important!</strong> If you need to specify multiple values for a single variable (e.g., a=b&amp;a=c&amp;a=d), use the table element.</td>
</tr>
</tbody>
</table>
Using the loadDocument Service

Use this element... | If you want to...
---|---

**Note:** When you use more than one element to submit data with loadDocument, *args* is appended first, *table* is appended second, and *string* is appended last.

The following procedure describes how to use the Pipeline Editor to assign data to *args* in a flow service.

**To assign name=value pairs in *args***

1. Select the loadDocument step in the Flow Editor, and then click the Pipeline tab.
2. In Service In, select *args*.
3. Click ☒ on the toolbar and select String.
4. Type the name portion of the first “name=value” pair, and press ENTER.
5. Click ✹ to make that element a child of *args*.
6. If you want to assign a hard-coded value to this element, do the following:
   1. Click ☂ on the toolbar to open the Input for dialog box.
   2. Type the value portion of the “name=value” pair. You do not need to URL-encode the value if you are using the GET method to submit data—loadDocument will do this at run time (e.g., it will convert “Global Sporting Goods” to “Global+Sporting+Goods”).
   3. Click OK.
7. If you want to assign a pipeline variable to this element, click ☐ and map the element to the appropriate variable in Pipeline In.
8. Repeat steps 3–7 for each “name=value” pair you want to submit.
Use this element... If you want to...

`table`

Specify your data in a two-column table of strings. When you specify data in this manner, `loadDocument` automatically appends the elements from `table` to `url` (if you are using the GET method) or inserts them in the body of the message (if you are using the POST method).

The table variable is similar to the `args` variable; however, `table` allows you to submit values that are not part of a "name=value" construction. For example, if your Internet resource expects a set of unnamed values (e.g., `http://www.rubicon.com/orders?GSG&40019&open`) instead of name=value pairs, you must specify those values in `table` (`args` only produces name=value pairs).

To submit data via the table element, create a row for each variable that you want `loadDocument` to submit, where:

- The contents of column 0 specify the name portion of the pair (if it is an unnamed variable, leave this cell blank), and...
- The contents of column 1 specify the value portion of the pair.

Note that `loadDocument` will automatically do the following:

- URL-encode values you specify.
- Insert the "&" character between each name=value pair that it constructs from `table` (for both POST and GET). You do not have to include this character in the values you specify.
- Insert the "?" character at the beginning of the query string it generates from `table` for the GET method. You do not have to include this character in the values you specify.

**Important!** When you use `args` and `table` to submit data with `loadDocument`, the service appends `args` first and `table` second. The service appends string last.

The following procedure describes how to use the Pipeline Editor to assign data to `table` in a flow service.
<table>
<thead>
<tr>
<th>Use this element...</th>
<th>If you want to...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>To assign name=value pairs in table</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Select the loadDocument step in the Flow Editor, and then click the Pipeline tab.</td>
</tr>
<tr>
<td>2</td>
<td>In Service In, select table.</td>
</tr>
<tr>
<td>3</td>
<td>If you want to assign a set of hard-coded values to table, do the following:</td>
</tr>
<tr>
<td>1</td>
<td>Click the toolbar to open the Input for dialog box.</td>
</tr>
<tr>
<td>2</td>
<td>Click twice to create two columns (0 and 1).</td>
</tr>
<tr>
<td>3</td>
<td>Click to create an empty row.</td>
</tr>
<tr>
<td>4</td>
<td>Select cell 0 and type the name of the name=value pair that you want to submit to the Internet resource in url. (If you are sending an unnamed variable, leave this cell empty.)</td>
</tr>
<tr>
<td>5</td>
<td>Select cell 1 and type the value of the name=value pair that you want to submit to the Internet resource in url. (If specifying an unnamed variable, type the variable’s value.)</td>
</tr>
<tr>
<td>6</td>
<td>Repeat steps 3 – 5 for each variable that you want to specify.</td>
</tr>
<tr>
<td>7</td>
<td>Click OK.</td>
</tr>
<tr>
<td>4</td>
<td>If you want to assign a pipeline variable to table, click and map the element to the appropriate variable in Pipeline In. (Note that the variable you map to table must be a two-column table of strings.)</td>
</tr>
<tr>
<td>Use this element...</td>
<td>If you want to...</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td><strong>string</strong></td>
<td>Specify your data as a single string of text. When you specify data in this manner, loadDocument automatically appends the specified string to url (if you are using the GET method) or inserts it into the body of the message (if you are using the POST method).</td>
</tr>
</tbody>
</table>

Keep the following points in mind when you use string to submit data with loadDocument.

- If you are specifying a set of name=value pairs, string must start with the first name in the set.
  
  **Example**  
  AccountNum=GSG-970414A

- You do not need to include the “?” symbol at the beginning of the string—loadDocument will automatically insert this symbol when it appends string to the contents of url.

- If you are using the GET method, be sure to format string as a valid query string—i.e., separate variables with the “&” character (as shown in Example 1) and URL-encode their data values (as shown in Example 2).
  
  **Example 1**  
  AccountNum=GSG-970414A&postalCode=22031

  **Example 2**  
  CompanyName=Global+Sporting+Goods

- If you are using the POST method, format the string exactly as you want it to appear in the body of the message.

**Important**! When you use more than one element to submit data with loadDocument, **args** is appended first, **table** is appended second, and **string** is appended last.
Use this element... If you want to...

*bytes*

Specify your data as an array of bytes that you want posted to the resource in url. You can use this variable when you need to send non-textual data to an Internet resource.

To assign a value to bytes in a flow service, use the Pipeline Editor to map the variable containing your byte array to bytes.

**Important!** loadDocument only uses the *bytes* when *method* is set to POST.

**Note:** When you use *bytes* and another element (*args, table, or string*) to submit data with loadDocument, the service appends the data from the *args, table, or string* element to *url*. The service appends *args* to *url* first, *table* second, and *string* last. The service encodes the data from the *bytes* element in the body of the post.

**The headers variable**

You use this variable if you want to specify the value of a header field in the HTTP request that loadDocument submits to the target server.

By default, loadDocument issues a standard set of header fields. (The values of some of these fields are configured when your SAP BC Server is installed. Consult your server administrator if you want to know your server’s default settings.) To use a header value other than the default, you can specify it in the *headers* variable.

**Important!** In most cases, the default headers that loadDocument uses are adequate. You should not add or change request header fields unless you are thoroughly familiar with the HTTP protocol.

If you want to assign specific values to header fields used by loadDocument, keep the following points in mind:

- When you specify the value of a header field, you override whatever default value SAP BC Server is configured to use for HTTP requests. For example, if you set the User-Agent header field to `BC/3.0`, the server uses that value instead of the default value specified by the `watt.net.userAgent` parameter.

- The loadDocument service automatically determines the value of the Content-Length header field. You cannot specify a value for Content-Length.

- Be aware that when you submit data using the *args* or *table* elements, loadDocument automatically sets the Content-Type header field to `application/x-www-form-urlencoded`. You cannot override this setting using the headers variable. If you want...
to explicitly specify a content type in headers, make sure to use the string or bytes element to submit your data, not args or table.

- Certain header fields are automatically derived from other input parameters assigned to loadDocument. For example, the Authorization header field is automatically derived from your auth parameter setting. Except for the Content-Length header field and the Content-Type header field (which, as described above, you cannot override when submitting data via args or table), a value that you specify in headers overrides the value that loadDocument might otherwise derive from other parameter settings.

- The loadDocument service does not validate data that you specify in headers. It simply passes it on to the target server in the request header. Make sure you specify header field names and their values correctly. For a complete list of valid request header fields, see http://www.w3.org for the latest HTTP specification published by the W3C.

To specify request headers in headers, create a string element for each header that you want to specify, where:

- The name of the element defines the name header field (e.g., User-Agent, If-Modified-Since, Mail_Address), and...

- The value of the element specifies the value you want assigned to that field.

The following procedure describes how to use the Pipeline Editor to assign values to headers in a flow service.

**To specify header fields in headers**

1. Select the loadDocument step in the Flow Editor, and then click the Pipeline tab.
2. In Service In, select headers.
3. Click on the toolbar and select String.
4. Type the name of the header field that you want to specify. (You do not need to type a colon after the field name—loadDocument will automatically insert the colon when it inserts this field into the request header.)
5. Click to make the element a child of headers.
6. If you want to hardcode the value of this element, do the following:
   1. Click on the toolbar to open the Input for dialog box.
   2. Type the value that you want loadDocument to assign to the headers field.
   3. Click OK.
7. If you want to assign a value from the pipeline to this element, click and map the element to the appropriate variable from Pipeline In.
8 Repeat steps 3–7 for each header field that you want to submit.

The encoding variable

You use this variable to specify the character set (e.g., ISO-8859-1) in which the returned document is encoded. The parser uses this value to properly read the text in the document.

You may set encoding to "autoDetect" or to the name of a specific, IANA-registered character set. When you set encoding to "autoDetect," loadDocument sets the character set based on whether the document contains XML or HTML. See the table below for defaults.

To explicitly specify a character set, specify the name of that character set in encoding. Valid character-set names are listed in the Internet Assigned Numbers Authority (IANA) character-set registry.

If you do not specify encoding, the parser assumes the following defaults depending on the type of document it receives:

<table>
<thead>
<tr>
<th>If the document type is...</th>
<th>encoding defaults to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML</td>
<td>ISO-8859-1</td>
</tr>
<tr>
<td>XML</td>
<td>UTF-8</td>
</tr>
</tbody>
</table>

The following procedure describes how to use the Pipeline Editor to assign values to encoding in a flow service.

To set the encoding value

1 Select the loadDocument step in the Flow Editor, and then click the Pipeline tab.
2 In Service In, select encoding.
3 If you want to hardcode the value of this element, do the following:
   1 Click [ ] on the toolbar to open the Input for dialog box.
   2 Type one of the following in the encoding field:

   Type...                             If you want the parser to...
   autoDetect                          Decode the document based on the document type. Assumes ISO-8859-1 for HTML documents and UTF-8 for XML documents.
   The name of a registered IANA character set. Decode the document based on a specified character set.

3 Click OK.
If you want to assign a value from the pipeline to this element, click \( \text{\textcopyright} \) and map the element to the appropriate variable from Pipeline In.

**The expandDTD variable**

You use this variable to specify whether you want the loadDocument service to expand instances of named-parameter entities in the returned document’s DTD. If you set `expandDTD` to true, `loadDocument` will process parameter-entity references, expanding them to their full definition. If you set `expandDTD` to false, `loadDocument` ignores parameter-entity references when it validates the returned document.

You may want or need to use this variable in cases when you have a syntactically-correct document that causes a parse error because it violates a definition in an external parameter-entity reference. By setting `expandDTD` to false, you can bypass the external definition so that `loadDocument` can parse the document successfully.

If you do not explicitly set `expandDTD`, `loadDocument` sets it to false.

**To set expandDTD in a flow service**

Do one of the following:

- Use the \( \text{\textcopyright} \) Set Value modifier in the Pipeline Editor and select the value of `expandDTD` from the list.
- OR –
- Use the \( \text{\textcopyright} \) Map modifier to assign a pipeline variable to `expandDTD`. The pipeline variable must set `expandDTD` to a value of true or false. (This value is case-sensitive.)

**The isXML variable**

You use this variable to specify whether the document you want to fetch contains HTML or XML. The server requires this value in order to parse the contents of the document correctly.

You may set `isXML` to any of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoDetect</td>
<td>Parses the document based on its <code>&lt;!DOCTYPE HTML &gt;</code> or <code>&lt;?xml version=...?&gt;</code> tag. If it cannot determine what type of document it has, it parses it as an HTML document.</td>
</tr>
<tr>
<td>false</td>
<td>Parses the returned document as HTML.</td>
</tr>
<tr>
<td>true</td>
<td>Parses the returned document as XML.</td>
</tr>
</tbody>
</table>
If you know what type of document loadDocument will receive, we suggest that you explicitly set isXML instead of using autoDetect. It will cut processing time, because the server will not have to examine the document to determine its type. The default value is autoDetect.

To set isXML in a flow service

Do one of the following:

- Use the Set Value modifier in the Pipeline Editor to select the value of isXML from the list.

   –OR–

- Use the Map modifier to assign a pipeline variable to isXML. The pipeline variable must set isXML to a value of autoDetect, true, or false. (This value is case-sensitive.)

The loadAs variable

You use this variable to specify the way in which loadDocument passes the parsed document to subsequent services that use it as input.

You may set loadAs to the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytes</td>
<td>Passes the parsed document as a byte array. You use this setting when the output from loadDocument will be used as input to a service that operates on whole documents (e.g., pub.web:queryDocument and pub.web:documentToRecord).</td>
</tr>
<tr>
<td>stream</td>
<td>Passes the parsed document as an input stream. You use this setting when the output from loadDocument will be used as input to a service that can incrementally process a document (e.g., pub.web:getNodeIterator).</td>
</tr>
</tbody>
</table>

If you do not explicitly set loadAs, loadDocument sets it to bytes.

To set loadAs in a flow service

Do one of the following:

- Use the Set Value modifier in the Pipeline Editor to select the value of loadAs from the list.

   –OR–
CHAPTER 16 Using the Load and Query Services

- Use the Map modifier to assign a pipeline variable to `loadAs`. The pipeline variable must set `loadAs` to a value of `bytes` or `stream`. (This value is case-sensitive.)

**The failOnHTTPError variable**

You use this variable to specify whether the service fails (throws an exception) if it receives an HTTP error as a response. For example, the server might return an HTTP 404 Not Found error for the requested URL. You may set `failOnHTTPError` to one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>The service does not fail based on the HTTP status code returned by the remote server. This is the default value.</td>
</tr>
<tr>
<td>true</td>
<td>The service fails if the remote server returns an HTTP error. The exception message contains the HTTP error code, which is determined by the W3C HTTP specification.</td>
</tr>
</tbody>
</table>

To set `failOnHTTPError` in a flow service

Do one of the following:

- Use the Set Value modifier in the Pipeline Editor to select the value of `failOnHTTPError` from the list.

  - OR-

- Use the Map modifier to assign a pipeline variable to `failOnHTTPError`. The pipeline variable must set `failOnHTTPError` to a value of true or false. (This value is case-sensitive.)

**Output Values from loadDocument**

If `loadDocument` executes successfully, it produces a single output, a `node`, which is a Document object containing the parsed HTML or XML document. You use `node` as input to other services such as `pub.xml:queryDocument` (to extract specific elements from the document) and `pub.web:documentToRecord` (to extract all of the document’s elements).

If `loadDocument` encounters an HTTP error during execution, the flow step fails and an exception is thrown if `failOnHTTPError` is set to `true`. See the section that follows.
Error Handling With loadDocument

If loadDocument fails, it returns an exception. If loadDocument fails because of an HTTP error and failOnHTTPError was set to true, then it returns an exception with the HTTP error code.
Common reasons for `loadDocument` to fail include:

- url does not begin with http: or https:
- `isXML` is true, but the document contains poorly-formed HTML.
- You have attempted to access a protected document without authorization (i.e., missing or incorrect user `auth/user` and/or `auth/pass` settings).
- `loadDocument` receives a syntactically incorrect document that cannot be parsed.
- An “Out of Memory” error.
- An HTTP error (e.g., the server cannot locate the requested document).
- Network timeout.
- `method` is not GET or POST.
- An HTTP error, such as “404 file not found” occurs.

### Using the `queryDocument` Service

You use the `pub.web:queryDocument` service to selectively extract information from a parsed HTML or XML document and assign that information to variables that can be mapped to other services.

When you use `queryDocument`, you extract information using either the XML Query Language (XQL) or the webMethods Query Language (WQL). Both languages allow you to address and select information from a node based on criteria that you specify in a query statement.

### Passing a Node to `queryDocument`

The `queryDocument` service takes a `node` as input. When you invoke `queryDocument` in a flow service, you must have a `node` variable (containing parsed XML or HTML) already in the pipeline for `queryDocument` to run against.

There are several ways in which you can produce a `node` for `queryDocument` to run against.

<table>
<thead>
<tr>
<th>If you want your service to...</th>
<th>Create a flow service that...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch an XML or HTML document from a specified URL.</td>
<td>1 Invokes <code>pub.web:loadDocument</code> (to get the document from the Web and transform it to a <code>node</code>), and then...</td>
</tr>
<tr>
<td></td>
<td>2 Invokes <code>pub.web:queryDocument</code>.</td>
</tr>
</tbody>
</table>
Using the queryDocument Service

The means that you use to obtain an HTML or XML document, convert it to a node, and pass it to the queryDocument service is a design decision that you must make before you begin building your flow.

Addressing Information in a Node

When SAP BC Server parses an HTML or XML document, it transforms the document into a tree-like structure containing the document’s HTML or XML elements. Once a document is parsed, you can address individual elements within it using an XQL or WQL query.

For example, in an HTML document, you could use the following query to extract the contents of the first paragraph in the document:

```plaintext
doc.p[0].text
```

In an XML document, you might use an XQL query that looks like this to extract the phone number from an element called supplierInfo in a SalesOrder document:

```plaintext
/SalesOrder[0]/supplierInfo[0]/phoneNum[0]/text()
```

The queryDocument service allows you to use either language to define a query. It also allows you to make this choice on a query-by-query basis—i.e., you can extract some information from the document using XQL and other information with WQL.

The language that you use is largely determined by personal choice and the needs of your application. Each language has a few specific features that the other does not. When you build your service, you can test your queries using either language to decide which provides the better results.

- For a description of WQL, see Appendix B, “webMethods Query Language” on page 571.
- For a description of XQL, see The XML Query Language (XQL), at http://www.w3.org/TandS/QL/QL98/pp/xql.html. (You can also refer to the XQL overview in <sapbc>\developer3\doc\ExampleXQL.txt.)
CHAPTER 16 Using the Load and Query Services

Working with a Sample Document

To help you build queries faster and more accurately, SAP BC Developer lets you generate queries by pointing to an element in a sample document. The sample document that you use should either be the actual document your service will execute against at runtime or a representative example of that document.

Note: Although you can define queries without loading a sample document, having one allows you to create your queries faster and more accurately.

To load a sample document into Developer, you execute your flow in “trace” mode (i.e., on the Test menu, click Trace). When you do this, Developer displays the parsed document on the Variables. (The Variables tab only appears when a queryDocument service is selected in the Flow Editor.)

The Document View tab displays the elements of a document

The Variables tab contains two additional tabs: the Document View tab and the Sample View tab.
The Document View Tab

The **Document View** tab shows the nodes (addressable components) that are contained in a document. It allows you to browse the contents of a document and generate queries by “pointing” to the node that you want to extract.

Developer uses the following symbols to denote different types of nodes in a document:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🍃</td>
<td>An HTML or XML element (e.g., <code>&lt;poNum&gt;…&lt;/poNum&gt;</code>). If an element contains additional nodes, the ⤴ symbol appears next to it. Click this symbol to view additional nodes within an element.</td>
</tr>
<tr>
<td>🏷</td>
<td>An element attribute (e.g., <code>name=&quot;Intro&quot;</code>).</td>
</tr>
<tr>
<td>🗨</td>
<td>A comment.</td>
</tr>
</tbody>
</table>

The Sample View Tab

The **Sample View** tab displays the result of a query. When you create a query, Developer dynamically executes that query against the sample document and displays the results on the **Sample View** tab.
The Sample View tab displays the results of the selected query

Adding queryDocument to a Flow Service

The following procedure describes the steps that you use to insert `pub.web:queryDocument` into a flow service. This procedure assumes that you have access to a sample document that you can load and display in Developer.

**Important!** The following procedure explains how to manually insert and specify the queryDocument service. If you are building a “load and query” sequence, you may want to use the wizard to build your flow instead. It allows you to quickly create a load and query service from a URL that you specify or by recording actions you make in an Web browser. For information about using the wizard to create a load and query service, see “Load and Query Shortcuts” on page 427.

To insert queryDocument into a flow service

1. In the Service Browser, select the flow in which you want to insert the queryDocument service.
2. In the Flow Editor, select the point where you want to invoke queryDocument, click in the Flow Editor toolbar and select `queryDocument` from the list. (If
queryDocument does not appear on the list, click Browse and select pub.web:queryDocument from the list next to Folder.)

3 Do one of the following to load your sample XML or HTML document into Developer:

   — If your flow passes a parsed XML or HTML document to queryDocument via a loadDocument or stringToDocument step, select the Trace command on the Test menu to execute the flow. (Make sure that the variables in the loadDocument or stringToDocument service are already correctly specified so that it will generate node successfully.)

   — If your flow expects the server to pass a parsed XML document to it, select the Send XML File command from the Test menu, and then do the following:

     1 In the Select Test Mode dialog box, select Trace and click OK.

     2 In the Select File dialog box, select your sample XML file and click Open.

4 When the trace finishes executing, click the Flow tab, select the queryDocument step in the Flow Editor, and then select Document View on the Variables tab.

5 Create a query for each piece of information that you want to extract from the document. See “Specifying a Query” on page 406 for detailed procedures.
If you use a namespace prefix to qualify element names in your query (e.g., /GSG:supplierInfo/name/text()), AND you want to map that prefix to a specific namespace in the document, use the following procedure to assign the prefix to the document’s namespace. (If you do not use namespace prefixes in your queries OR the prefixes you use match the ones used in the document, you can skip this step.)

1. Click the **Pipeline** tab.

2. Select the **nsDecls** variable in Service In and click on the toolbar to open the **Input for** dialog box.

3. Click and specify the following:

<table>
<thead>
<tr>
<th>In this field...</th>
<th>Specify...</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix</td>
<td>The prefix you use to represent a namespace in your queries.</td>
</tr>
<tr>
<td>uri</td>
<td>The namespace represented by this prefix. (Namespaces are named using URL notation.)</td>
</tr>
</tbody>
</table>

**Note:** When nsDecls is a single IData object, the key identifies the prefix, and the value identifies the URL. When it is a Vector, nsDecls contains a collection of IData objects as defined above. When it is a String[2], the first column is the prefix and the second is the URI.

4. Click **OK**.

5. Repeat steps 3 and 4 for each namespace prefix that you need to define. When you are finished, click **OK** to close the **Input for** dialog box.

### Specifying a Query

You use the **Variables** tab to define the queries that queryDocument will execute to populate a set of variables. The left side of the tab displays the list of variables that you want queryDocument to populate. The right side of the tab contains the controls you use to build a query that will populate a variable.

To examine and/or edit the query for a particular variable, you select the variable from the list on the left side and view and/or adjust the settings (Query, Type, Allow Null, and so forth) on the right.
The Variables tab defines the variables that queryDocument will produce

There are two ways you can define a variable (and the query that will populate it) for queryDocument:

- You can manually type the query into the Variables tab.
- OR -
- You can create the query by “pointing” to the node you want to extract.

Specifying a Query Manually

If you are familiar with the contents and structure of the document you want to query, you can manually build a query to extract information from it. When you create a query in this way, you do not need to load a sample document into Developer (however, you may eventually want to load one to test your queries and verify that they work correctly).

The following procedure describes how to manually create an input variable for the queryDocument service and assign a query to it.
To manually create a variable for a queryDocument service

1. If the toolbar on the Variables tab is not active, click anywhere in the variable's list box (the white area on the left side of the tab) to activate it.

2. Click on the toolbar to create a new variable in the list.

3. Type a name for the variable and press ENTER.

4. Set the following options:

<table>
<thead>
<tr>
<th>Set this option...</th>
<th>To specify...</th>
</tr>
</thead>
<tbody>
<tr>
<td>WQL/XQL</td>
<td>The language in which you will write the query.</td>
</tr>
<tr>
<td></td>
<td>Set... To...</td>
</tr>
<tr>
<td></td>
<td>WQL Use the webMethods Query Language.</td>
</tr>
<tr>
<td></td>
<td>XQL Use the XML Query Language.</td>
</tr>
<tr>
<td>Allow Nulls</td>
<td>Whether a null or non-null result should cause the service to fail at run time.</td>
</tr>
<tr>
<td></td>
<td>Set... If queryDocument should...</td>
</tr>
<tr>
<td></td>
<td>Allow Null Accept both null and non-null results.</td>
</tr>
<tr>
<td></td>
<td>Fail if Null Issue an exception if the query produces a null result.</td>
</tr>
<tr>
<td></td>
<td>Fail if Not Null Issue an exception if the query produces anything but a null result.</td>
</tr>
<tr>
<td>Type</td>
<td>The data type of the variable in which you want the query result stored</td>
</tr>
</tbody>
</table>

5. In the Query field, type your query and then press ENTER.

Important! When querying an XML document, make sure to type node names exactly as they are specified in the document. For XML, node names are case-sensitive! (HTML node names are not.)

Developer automatically executes the query when you press ENTER. If you have a sample document loaded, you can click the Sample View tab to display the result.
Specifying a Query by Pointing to a Node

To create queries quickly and accurately, you can select a node in the Document View tab and have Developer automatically build a query that selects the text for that element. After Developer builds the query, you can use it “as is” or edit it to suit your needs.

Important! To use the following procedure, you must load the source document in the Document View tab. If you have not already done this, run the Test > Trace command (if queryDocument receives the node from a loadDocument or stringToDocument service) or Test > Send XML File (if queryDocument receives input directly from the server) before you begin.

Important! The following procedure produces a query in Developer's default query language. To view and/or change your current default language, on the Edit menu, click Preferences, click the General tab, and adjust the QueryDocument setting.

To create a query from a node in the Document View tab

1. On the Document View tab, select the node for which you want to generate a query.
2. Right-click and select Create New Variable from Node.
3. Type a name for the variable that will hold the results of the query, and press ENTER.
4. Set the following options:

<table>
<thead>
<tr>
<th>Set this option</th>
<th>To specify...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow Nulls</td>
<td>What queryDocument should do if the query produces a null result at run time.</td>
</tr>
<tr>
<td></td>
<td>If queryDocument should...</td>
</tr>
<tr>
<td></td>
<td>Set...</td>
</tr>
<tr>
<td>Allow Null</td>
<td>Accept both null and non-null results.</td>
</tr>
<tr>
<td>Fail if Null</td>
<td>Issue an exception if the query produces a null result.</td>
</tr>
<tr>
<td>Fail if Not Null</td>
<td>Issue an exception if the query produces anything but a null result.</td>
</tr>
</tbody>
</table>

5. Type | The data type of the variable in which you want the query result stored. |
   | The data type of the variable in which you want the query result stored. |

5. If necessary, edit the query in the Query field. Press ENTER when you finish editing to re-execute the query. To view the results, click the View Sample tab. For additional information about creating queries, see “Using WQL and XQL” below.
Using WQL and XQL

WQL (webMethods Query Language) is a language that is used to retrieve information from an HTML or XML document. WQL uses an object reference and an object property in order to extract document elements from an HTML or XML document. In its simplest form, the syntax is Objectreference.Objectproperty. For example, order[].text means: Select the text of all order elements in a particular context (for example, within an XML file).

XQL (XML Query Language) is a language that is used to retrieve information from an XML document. XQL uses simple directory notation. For example, order/lineItem means: Select all lineItem elements in all order elements in a particular context (for example, within an XML file). You can also use Boolean logic to filter out elements, select all nodes of a particular value, select all nodes with nodes in particular ranges, and so forth.

Using WQL in a Query

A WQL query consists of one or more indexed element arrays and an object property. For example, all of a document’s snowboard product elements may be collected into an array called snowboard[], its product names may be collected into an array called name[], its product prices may be collected into an array called price[], and so forth.

Partial XML Document

```xml
<?xml version="1.0"?>
<root>
  <snowboard>
    <name>Globe</name>
    <type>Freeride/Freestyle</type>
    <sku>A</sku>
    <length>149</length>
    <width>28.5</width>
    <price>279</price>
    <availability>true</availability>
  </snowboard>
  <snowboard>
    <name>Pro Series</name>
    <type>Freeride/Powder/Extreme</type>
    <sku>B</sku>
    <length>156</length>
    <width>28.9</width>
    <price>399</price>
    <availability>false</availability>
  </snowboard>
</root>
```
The arrays are indexed numerically, beginning with 0. To extract a particular element, you specify its address within the array of which it is a member. For example, the following is the WQL query for the first snowboard element in the previous example:

```
doc.snowboard[0].text
```

An object property specifies the type of information that you want to extract from that element. In the WQL example above, the .text suffix specifies that you want to extract the text contained within `doc.snowboard[0]`. This suffix is the object property. This property can either be one of webMethods predefined properties or one of the element's attributes (e.g., WIDTH, ALIGN, HREF).

**webMethods Predefined Properties**

WQL provides a set of predefined properties that you can use to extract specific kinds of information from an element. For example, to get the text contained in the `<name>` element shown below:

```
<name>Globe</name>
```

You use the .text property as follows:

```
doc.name[0].text
```

The following table describes the set of predefined properties in the webMethods Query Language. Note that each property has two names. This provides an alternative name you can use in cases where an element has an attribute with the same name as the WQL property (the attribute name takes precedence over a WQL property name when the two names are the same).

<table>
<thead>
<tr>
<th>Use this property...</th>
<th>To get...</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td>The text contained within the specified element. If the specified element contains child elements, the contents of all of those elements are extracted too.</td>
</tr>
</tbody>
</table>
| .txt                 | Example doc.name[0].text
|                      | Result "Globe"
| .value               | The value contained within the specified element. If the specified element contains child elements, the values for all of those elements are extracted too. |
| .val                 | Example doc.name[0].value
|                      | Result "Globe"
| .source              | The XML or HTML source code for the specified element. |
| .src                 | Example doc.name[0].source
|                      | Result <name>Globe</name>
Element Attributes

In addition to the predefined properties provided by the webMethods Query Language, you can also specify an attribute name as a property. When you specify an attribute name as a property, the query returns the value of that attribute. For example, in the following XML element:

```xml
<part name="Widget" sku="123456" quantityonhand="5000">
</part>
```

You would use the following query:

```
doc.part[0].sku
```

To extract the following from the SKU attribute:

"123456"

Specifying Multiple Elements

To extract a single item from a document, specify a query that extracts a single member of that array as shown in the following examples:

<table>
<thead>
<tr>
<th>This reference...</th>
<th>Returns data from...</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc.snowboard[0].text</td>
<td>The first snowboard element in the document.</td>
</tr>
<tr>
<td>doc.snowboard[1].text</td>
<td>The second snowboard element in the document.</td>
</tr>
<tr>
<td>doc.snowboard[5].text</td>
<td>The sixth snowboard element in the document.</td>
</tr>
</tbody>
</table>

To extract multiple members from an object array, you can specify those objects using special index notation as shown in the following examples.
This reference... | Returns data from...
---|---
`doc.snowboard[ ].text` | All snowboard elements in the document.
`doc.snowboard[1-10].text` | Snowboard elements 1 through 10 in the document.
`doc.snowboard[1-end]` | Snowboard elements 1 to the end of the document.
`doc.snowboard[1,3,7]` | Snowboard elements 1, 3, and 7.
`doc.snowboard[1-3,5-7]` | Snowboard elements 1, 2, 3, and 5, 6, 7.
`doc.snowboard[2+2]` | Snowboard elements 2, 4, 6, 8...

For a complete description of the index value options, see Appendix B, “webMethods Query Language” on page 571.

**Nested Element References**

You can specify an element relative to any of the elements it is contained within (that is, relative to its parent containers). For example, the second product name element (underscored below) in the following XML fragment:

```
<snowboard>
  <name>Vert</name>
  <type>Freestyle/Pipe/Park</type>
  <sku>D</sku>
  <length>154</length>
  <width>25</width>
  <price>479</price>
  <availability>true</availability>
</snowboard>

<snowboard>
  <name>Trauma</name>
  <type>Freestyle/Pipe/Park</type>
  <sku>E</sku>
  <length>153</length>
  <width>29.4</width>
  <price>367</price>
  <availability>true</availability>
</snowboard>
```

Can be referenced in either of the following ways:

- `doc.name[1].text`
- `doc.snowboard[1].name[0].text`

The index value varies, depending on which parent elements you include in the reference. The first container in a WQL query is always `doc`, which references a node.
The Scope of a WQL Query

Nested object references also define the scope of a query. Each of the following examples references an element called name[0], but by nesting elements in the query, you constrain the scope of that query.

For this query... The scope is...

```
doc.name[0].text
```
The entire document.

```
doc.snowboard[0].name[0].text
```
The first snowboard element in the document.

```
doc.bitterrootboards[2].snowboard[0].name[0].text
```
The first snowboard element within the third bitterrootboards element in the document.

Referencing Data in Tables

To retrieve the entire contents of a table within an HTML document, specify that table's index value, and use [] for the rows and columns in that table.

```
doc.table[0].tr[].td[].text
```

Besides table references, other queries can create a table as a result. For example, the following query gets all bolded text from all list items in an HTML document:

```
doc.ul[0].li[].b[].text
```

Its result may be one or more bolded items from each list item. To hold this type of output, the variable must also be a String Table.

Note: References that combine more than two arrays are collapsed into two-dimensional tables.

Using Search Strings as Index Values

You can use pattern-matching strings instead of index numbers to select members of an element array. This type of query returns data from those elements whose text property matches the specified pattern-matching string. For example, the following query:

```
doc.bitterrootboards["*Pro Series*"].text
```

Returns every bitterrootboards element that contains the string “Pro Series” anywhere within it.

Wildcard Symbols

Pattern-matching strings used in place of an index value can include the following wildcards:
Use this symbol... To...

*     Match any sequence of zero or more characters.
?     Match any single character.
%     Match a single word (i.e., a sequence of non-whitespace characters).
\    Escape the above metacharacters.

**Example** You want to return all bitterrootboards elements that contain the character “?”. You use the following query:

```
doc.bitterrootboards["\?"].text
```

**Note:** Match strings are case-sensitive.

**Examples**

<table>
<thead>
<tr>
<th>This reference...</th>
<th>Returns...</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc.name[&quot;<em>fourth</em>&quot;].text</td>
<td>The text from all name elements containing the string “fourth”.</td>
</tr>
<tr>
<td>doc.name[&quot;<em>fourth</em>&quot;].src</td>
<td>The source code from all name elements containing the string “fourth”.</td>
</tr>
<tr>
<td>doc.tr[&quot;199? *&quot;].idx</td>
<td>The index of every row that contains a string beginning with “199” followed by any character and a space.</td>
</tr>
<tr>
<td>doc.table[&quot;<em>sku</em>&quot;].tr[].text</td>
<td>The text from every row of any table containing the string “sku”.</td>
</tr>
<tr>
<td>doc.tr[&quot;<em>Green</em>&quot;].td[1].idx</td>
<td>The text from the second cell of any table row containing the string “Green”.</td>
</tr>
<tr>
<td>doc.table[1].tr[&quot;<em>Green</em>&quot;].td[1].text</td>
<td>The text from the second cell of any row containing the string “Green” in the second table.</td>
</tr>
<tr>
<td>doc.tr[&quot;*H????<em>Red</em>&quot;].td[1].text</td>
<td>The text from the second cell of any row that contains a string beginning with “H” separated by at least 4 characters from a string that begins with “Red”.</td>
</tr>
</tbody>
</table>
Regular Expressions as Index Values

As an alternative to using webMethods standard pattern-matching syntax described above, you can use a regular expression to specify a pattern-matching string. When you use this method, you enclose the regular expression in “/” characters as shown in the following example:

```java
doc.bitterrootboards[/*Pro Series*/].text
```

This example will return every bitterrootboards element that contains the string “Pro Series” anywhere within it. For information about regular expressions, see Appendix D, “Regular Expressions” on page 585.

Attribute Matching

By default, match strings are compared with the .text property of the specified elements; however, you can match on other properties by specifying that property and a match string as shown in the following example.

```java
doc.a(HREF='*webmethods.com*').src
```

This example extracts all hypertext links to “*webmethods.com*”.

**Important!** When you want to search a specific property, you must enclose the property assignment statement in *parentheses*, not brackets.

Matching on the Name Attribute

Elements that contain a NAME attribute can be referenced directly by name. For example, to get the text from all elements having the name “sku”, you would use the following reference:

```java
doc.sku.text
```

To get the text from all table cells having a width value of 8% from all elements named “sku”, you would use the following:

```java
doc.sku.td(width='8%').text
```

Using Masks to Extract Data

When you specify a query, that query returns the complete contents of the specified property. For example, the following query:

```java
doc.snowboard[1].text
```

Returns all of the text from the specified snowboard element. If you want to extract only a portion of the returned text, you can apply a mask to the text property to eliminate unwanted information. For example, the following object reference:

```java
doc.snowboard[1].text[‘&list*’]
```

Eliminates the word “list” and every character that follows it.
The following table describes the symbols you use to construct a mask:

<table>
<thead>
<tr>
<th>Use this symbol...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>Eliminate a single character from the result.</td>
</tr>
<tr>
<td>*</td>
<td>Eliminate a string containing any characters.</td>
</tr>
<tr>
<td>%</td>
<td>Eliminate a single word (i.e., a sequence of non-whitespace characters).</td>
</tr>
<tr>
<td>$</td>
<td>Collect a single word and include it in the result.</td>
</tr>
<tr>
<td>&amp;</td>
<td>Collect a string containing any characters and include that string in the result.</td>
</tr>
<tr>
<td>\</td>
<td>Escape the above metacharacters.</td>
</tr>
</tbody>
</table>

**Example** You want to return the text of all snowboard elements in the document, eliminating all asterisks (*) from the results. You use the following query:

```plaintext
doc.snowboard[].text['*']
```

A mask pattern is matched against the text of an object reference until the pattern is exhausted, and any remaining characters will be included in the result.

**Examples**

<table>
<thead>
<tr>
<th>This mask...</th>
<th>Performs the following...</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text['&amp;list*']</td>
<td>Keeps all characters up to (but excluding) the string “list”, and then eliminates the rest.</td>
</tr>
<tr>
<td>.text['%%&amp;']</td>
<td>Eliminates the first three words in the result and keeps the remaining characters.</td>
</tr>
<tr>
<td>.text['<em>page&amp;how</em>']</td>
<td>Eliminates all characters up to (and including) the string “page”; keeps all characters up to (but excluding) the string “how”; and eliminates the rest.</td>
</tr>
</tbody>
</table>

**Regular Expressions in a Mask**

As an alternative to using webMethods standard mask syntax described above, you can use a regular expression to specify a mask. When you use this method, you enclose the regular expression in “/” characters as shown in the following example:

```plaintext
doc.snowboard[1].text[/list*/]
```

This example will eliminate the word “list” and every character that follows it from the result. For information about regular expressions, see Appendix D, “Regular Expressions” on page 585.
Using Lines to Select Data

If an element contains <BR> tags, you can address each line of that element individually using the line[] array. For example, to reference the third line in the following paragraph:

```javascript
<P>
The current balance for customer account<BR>1001-233<BR>is $14,009.50<BR>as of 14:04pm 10/16/98.<BR></P>
```

You can use the following query:

```
doc.p[0].line[2].text
```

You can also reference lines of text within a <PRE> element in the same way. For example, to get the second line in the following segment of preformatted text:

```javascript
<PRE>
ITEM DESC QTY UPRICE  TOTAL
06-5449 1” ITC Casing 250   .43 $ 107.50
010-13 #4 PVC Line 50 23.50 $1175.00
00901R Tak Connector 25  3.50 $  87.50
</PRE>
```

You use the following query:

```
doc.pre[0].line[1].text
```

Like other elements, you can use numeric ranges or pattern-matching strings as the index to the line array.

Examples

<table>
<thead>
<tr>
<th>This reference...</th>
<th>Returns data from...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>doc.p[0].line[].text</code></td>
<td>All lines of text in the first paragraph.</td>
</tr>
<tr>
<td><code>doc.p[0].line[1].text</code></td>
<td>The second line of text in the first paragraph.</td>
</tr>
<tr>
<td><code>doc.p[0].line[&quot;demo&quot;].text</code></td>
<td>All lines of text in the first paragraph that contains the string “demo”.</td>
</tr>
<tr>
<td><code>doc.pre[0].line[2,8].text</code></td>
<td>Lines 3 and 9 from the first preformatted text element in the document.</td>
</tr>
<tr>
<td><code>doc.pre[0].line[0+5].text</code></td>
<td>Every fifth line of text from the first preformatted text element in the document.</td>
</tr>
</tbody>
</table>
Using Character Ranges to Extract Data

You can use a special type of property mask to return a specific range of characters from a result. For example, the following query extracts only the first 10 characters from the second name element:

\[ \text{doc.name[1].text[0-9]} \]

The combination of character ranges and line array allows you to create very complex queries and gives you tremendous flexibility in getting specific data from text objects. For example, the following query:

\[ \text{doc.pre[0].line[*connector*].text[43-50]} \]

Gets the text from columns 43 through 50 of any line in the first preformatted element containing the string “connector.”

Using XQL in a Query

Data parsed from an XML document can be accessed with a XQL query, which consists of basic URI syntax that specifies elements in the XML tree. For example, the following specifies the collection of poNum elements within SalesOrder elements, assuming that the root element of the document is SalesOrder:

\[ /\text{SalesOrder/poNum} \]

The collection of all elements with a certain tag name is expressed using the tag name itself. This can be further qualified by indicating that the scope is the current context “/”, but the current context is assumed and need not be noted explicitly.

Example

To select all poNum elements, the following are equivalent:

\[ /\text{poNum} \]
\[ \text{poNum} \]

Specifying a Method in XQL

XQL provides a set of predefined methods that you can use to extract specific kinds of information from an element. For example, to get the text contained in the first \(<\text{poNum}>\) element in the following example:

\[ <\text{SalesOrder}> \]
\[ <\text{poNum}>GSG-99401088</poNum> \]

You use the \text{text()} method as follows:

\[ /\text{SalesOrder/poNum[0]/text()} \]

The following table describes the set of predefined methods in XML Query Language.
## Use this method... To get...

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text()</td>
<td>The text contained within the specified element, minus any white space</td>
</tr>
<tr>
<td></td>
<td>such as spaces, tabs, and line breaks. If the specified element</td>
</tr>
<tr>
<td></td>
<td>contains child elements, the contents of all of those elements are</td>
</tr>
<tr>
<td></td>
<td>extracted too.</td>
</tr>
<tr>
<td>rawtext()</td>
<td>The text contained within the specified element, including any white space.</td>
</tr>
<tr>
<td>value()</td>
<td>The value contained within the specified element. If the specified element</td>
</tr>
<tr>
<td></td>
<td>contains child elements, the values for all of those elements are</td>
</tr>
<tr>
<td></td>
<td>extracted too.</td>
</tr>
<tr>
<td>float()</td>
<td>The value contained within the specified element, returned as a floating</td>
</tr>
<tr>
<td></td>
<td>decimal. This method is useful in comparisons. See “Specifying Elements</td>
</tr>
<tr>
<td></td>
<td>with Comparison Operators” on page 425.</td>
</tr>
<tr>
<td>integer()</td>
<td>The value contained within the specified element, returned as an integer.</td>
</tr>
<tr>
<td></td>
<td>This method is useful in comparisons. See “Specifying Elements with</td>
</tr>
<tr>
<td></td>
<td>Comparison Operators” on page 425.</td>
</tr>
<tr>
<td>index()</td>
<td>The index number of the node within the parent.</td>
</tr>
<tr>
<td>nodeType()</td>
<td>The number that indicates the type of node.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number</th>
<th>Node Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Element</td>
</tr>
<tr>
<td>2</td>
<td>Attribute</td>
</tr>
<tr>
<td>3</td>
<td>Text</td>
</tr>
<tr>
<td>7</td>
<td>PI</td>
</tr>
<tr>
<td>8</td>
<td>Comment</td>
</tr>
<tr>
<td>9</td>
<td>Document</td>
</tr>
</tbody>
</table>
Use this method... | To get...
---|---
namespace() | The URI for the namespace of the node.

*Example* /SalesOrder/poNum[0]/namespace()

baseName() | The name portion of the node, excluding the prefix.

*Example* /SalesOrder/poNum[0]/baseName()
*Result* poNum

prefix() | The prefix for the node.

*Example* /SalesOrder/poNum[0]/prefix()
*Result* AcmeCorp (the variable in the xmlns statement of the document)

end() | “True” if the node is the last in the collection, relative to the parent node. “Null” if the index value is out of range.

*Example* /SalesOrder/poNum[0]/end()
*Result* true

*Example* /SalesOrder/poNum[10]/end()
*Result* null

**Element Attributes**

In addition to the predefined methods provided by XQL, you can also specify an *attribute name* in a query. When you specify the “@” character and an attribute name, the query returns the value of that attribute. For example, in the following XML element:

```
<part name="Widget" sku="123456" quantityonhand="5000">
</part>
```

You would use the following query:

```
part[0]/@sku
```

To extract the following from the SKU attribute:

"123456"

If you want to extract all values of all attributes within an element, you use the at (@) and asterisk (*) wildcard characters. This is useful if you want to treat attributes as fields in a record. For example, in the following XML element:

```
<part name="Widget" sku="123456" quantityonhand="5000">
</part>
```

You would use the following query:

```
part[0]/@*
```

To extract the following from the part element:

"Widget" "123456" "5000"
Important! Attributes cannot have path operators nor indices appended to them in a query. Such queries will result in syntax errors.

Specifying Multiple Elements

To extract multiple elements from an XML document, you can specify those nodes using special index notation as shown in the following examples. If you do not specify an index number, the query returns all elements.

This query... Returns data from...

\text{vendorNum/text( )} \quad \text{All vendorNum elements in the document.}
\text{vendorNum[0 \text{ to } 2]/text( )} \quad \text{The first three vendorNum elements in the document.}
\text{vendorNum[1,3,7]/text( )} \quad \text{vendorNum elements 1, 3, and 7.}

Nested Element References

You can specify an element relative to any of the elements it is contained within (that is, relative to its parent containers). For example, the second companyName element (underscored below) in the following XML fragment:
Partial XML Document

```xml
<buyerInfo>
  <companyName>Global Sporting Goods</companyName>
  <accountNum>GSG-970414-0A</accountNum>
  <phoneNum>(216) 741-7566</phoneNum>
  <faxNum>(216) 741-7533</faxNum>
  <addr>
    <streetAddr>10211 Brookpark Rd</streetAddr>
    <city>Cleveland</city>
    <state>OH</state>
    <postalCode>22130</postalCode>
  </addr>
  <purchEmail>buyers@GSG.com</purchEmail>
  <purchRep>Caroline Wielman</purchRep>
</buyerInfo>

<supplierInfo>
  <companyName>Bitterroot Boards, LLC</companyName>
  <vendorNum>8444 99731 Q64500</vendorNum>
  <phoneNum>(406) 721-5000</phoneNum>
  <faxNum>(406) 721-5001</faxNum>
  <addr>
    <streetAddr>Suite 441</streetAddr>
    <streetAddr>1290 Antelope Dr</streetAddr>
    <city>Missoula</city>
    <state>MT</state>
    <postalCode>59801 1290</postalCode>
  </addr>
  <salesEmail>orders@GSG.com</salesEmail>
  <salesRep>Marc Norgaard</salesRep>
</supplierInfo>

<order>
  <lineItem>
    <stockNum>BK-XS160</stockNum>
    <desc>Extreme Spline 160 Snowboard- Black</desc>
    <qty>10</qty>
    <uPrice>149.00</uPrice>
    <extPrice>14900.00</extPrice>
  </lineItem>
</order>
```

Can be referenced in either of the following ways:

```xml
//companyName[1]/text()
//supplierInfo[0]/companyName[0]/text()
```

The index value varies, depending on which parent elements you include in the reference.

The first several characters are path operators. You use these operators to “drill-down” levels in the document you are querying. The “/” operator before an element indicates the immediate child; the “//” operator indicates any descendant from the current context. See the examples below.
### Using Wildcard Characters for Child Elements

You can specify all child elements of a parent element by using an asterisk (*) character. See the examples below.

<table>
<thead>
<tr>
<th>This query...</th>
<th>Returns data from...</th>
</tr>
</thead>
<tbody>
<tr>
<td>//order/*</td>
<td>All child elements of all order elements in the document.</td>
</tr>
<tr>
<td>//order/*/stockNum</td>
<td>All stockNum elements that are grandchildren of any order elements in the document.</td>
</tr>
</tbody>
</table>

### Filtering Elements that Contain a Specific Child Element

You can specify elements that contain a particular child element. See the examples below.

<table>
<thead>
<tr>
<th>This query...</th>
<th>Returns data from...</th>
</tr>
</thead>
<tbody>
<tr>
<td>//order[extPrice]</td>
<td>All order elements that have an extPrice child element.</td>
</tr>
<tr>
<td>//order[extPrice][uPrice]</td>
<td>All order elements that have at least one extPrice element and Price child element.</td>
</tr>
</tbody>
</table>

**Note:** You can also filter elements using Boolean logic. For details, see “Specifying Elements with Boolean Expressions” on page 425.

### Using Search Strings as Index Values

You can use pattern-matching strings instead of index numbers to select members of an element array. This type of query returns data from those elements whose text property matches the specified pattern-matching string. For example, the following query returns every companyName element that contains “Bitterroot Boards, LLC”:

```xml
//companyName[text()='Bitterroot Boards, LLC']
```

You can also use the “/” character to specify the text of a child element. For example, the following query returns every supplierInfo element that contains “Bitterroot Boards, LLC” in the companyName elements in the document:

```xml
//supplierInfo[companyName/text()='Bitterroot Boards, LLC']
```
If you want to return all elements that contain a certain string in any form, you can use the following `regex(pattern)` function. For example, the following query returns the text of all AcctNum elements that contain the string “0100” anywhere within them:

```
//AcctNum[regex("0100")]/text()
```

**Specifying Elements with Particular Namespaces**

If you want to find elements that contain specific namespace information, you use the colon “:” character. For example, the following query:

```
//buyerInfo:companyName[]
```

Returns every companyName element with the namespace “buyerInfo” anywhere in the document.

**Specifying Elements with Boolean Expressions**

Within queries, you can use Boolean expressions to filter out certain elements, select all elements of a particular value, and select all elements in particular ranges. Boolean expressions use `$and$, `$or$, and `$not$; however, use of the “$” character is optional in Developer. See the following examples.

<table>
<thead>
<tr>
<th>This query...</th>
<th>Returns data from...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>//order[extPrice and uPrice]</code></td>
<td>All order elements that have extPrice and Price elements.</td>
</tr>
<tr>
<td><code>//order[(extPrice or uPrice) and stockNum]</code></td>
<td>All order elements that have at least one extPrice or uPrice element and at least one stockNum element.</td>
</tr>
<tr>
<td></td>
<td>Note that you can use parentheses to set operation precedence.</td>
</tr>
<tr>
<td><code>//order[not uPrice]</code></td>
<td>All order elements that do not contain uPrice elements.</td>
</tr>
<tr>
<td><code>//order[extPrice and not uPrice]</code></td>
<td>All order elements that contain at least one extPrice element and contain no uPrice elements.</td>
</tr>
</tbody>
</table>

**Specifying Elements with Comparison Operators**

You can also use comparison operators to filter out certain elements, select all elements of a particular value, and select all elements in particular ranges. Comparison operators include “=”, “!=” (not equal to), “<”, and “>”. See the following examples.
This query... | Returns data from...
---|---
//vendorNum[.='8444'] | All vendorNum elements in the document that equal 8444.
//vendorNum[.'!=8444'] | All vendorNum elements in the document that do not equal 8444.
//vendorNum[.'<8444'] | All vendorNum elements in the document that are less than 8444.
//vendorNum[.'>8444'] | All vendorNum elements in the document that are greater than 8444.
//supplierInfo[vendorNum='8444'] | All supplierInfo elements that contain a vendorNum element that equals 8444.

Note: For the purposes of comparison, value( ) is implied if omitted. Keep in mind that the value is treated as a string unless you specify otherwise. To compare numeric values, use the integer( ) or float( ) method on both sides of the expression. For example, //uPrice[float(.)<float('9.99')].

In addition, when an element reference is used in a comparison, you must only specify one element for comparison. For example, //LineItem[qty>/PO/AcctNum[]] is not a valid query, because all AcctNum elements are specified for comparison.

Quick Reference: WQL vs. XQL

Use the following table as a guideline for WQL and XQL syntax. For details, see Appendix B, “webMethods Query Language” on page 571.

<table>
<thead>
<tr>
<th>If you want...</th>
<th>Use this WQL statement...</th>
<th>Or this XQL statement...</th>
</tr>
</thead>
<tbody>
<tr>
<td>All <strong>product</strong> elements in the document</td>
<td>product[ ]</td>
<td>//product</td>
</tr>
<tr>
<td>The text of the first <strong>name</strong> element within the first <strong>product</strong> element</td>
<td>product[0].name[0].text</td>
<td>//product[0]/name[0]/text( )</td>
</tr>
<tr>
<td>All <strong>product</strong> elements with a value of TextContent</td>
<td>product[&quot;TextContent&quot;]</td>
<td>//product=&quot;TextContent&quot;</td>
</tr>
</tbody>
</table>
Output from queryDocument

The queryDocument service produces one output variable for each query that you specify in the Variables tab.

Handling Errors from queryDocument

If queryDocument fails, it throws a server exception. Common reasons for queryDocument to fail include:

- A variable that has no query string assigned to it.
- A syntax error in a query string.
- A query fails the “Allows Nulls” test.
- The node variable does not exist or it is null.

Load and Query Shortcuts

If you need to create a load and query service (a flow that invokes a loadDocument service and a queryDocument service), you can use the wizard to build it for you. It can quickly generate this type of service based on a URL that you specify or by recording your actions in an Internet browser.

Generating Default Variables

Besides speed and ease-of-use, the wizard offers another benefit—it can produce a set of default variables to extract frequently-used elements (e.g., <p>, <table>, <li>, and so forth) from an HTML document. During development, the default variables are useful ways to locate a particular piece of information in a document and ascertain its address. For example, if you want to select information from a particular table in a document, you can browse the Tables variable to obtain the index for that table.
The following table describes the set of default variables the wizard can produce when it builds a load and query service.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Extracts the text from the document’s <code>&lt;title&gt;</code>…<code>&lt;/title&gt;</code> element.</td>
</tr>
<tr>
<td>Anchors</td>
<td>Collects the values of all HREF attributes in the document.</td>
</tr>
<tr>
<td>Paragraphs</td>
<td>Collects the text from all <code>&lt;p&gt;</code> elements in the document and stores it in a String Table.</td>
</tr>
<tr>
<td>Tables</td>
<td>Collects the text from all <code>&lt;table&gt;</code> elements in the document and stores it in a String Table.</td>
</tr>
<tr>
<td>Forms</td>
<td>Collects information about each <code>&lt;form&gt;</code> element in the document and stores it in a Record List.</td>
</tr>
<tr>
<td>Lists</td>
<td>Collects the text from all <code>&lt;li&gt;</code> elements (from ordered and unordered lists) in the document and stores it in a String Table.</td>
</tr>
</tbody>
</table>

⚠️ Important! Default variables are not usually used by the service you build (although you may use them if they suit your purpose). They are provided primarily as guides for locating and examining information in a document during the development stage. You usually delete the default variables after you finalize the variable list for queryDocument.

Creating a Load and Query Service from a URL

The following procedure describes how to use the wizard to create a load-and-query service for a URL that you specify. When you use this method, Developer generates a flow service made up of a loadDocument service followed by a queryDocument service, and automatically configures the input variables for these services based on options that you specify in the wizard.

After the wizard generates the load-and-query service, you can do any of the following:

- Use the flow “as is.”
- Edit the input values for the loadDocument and/or queryDocument service.
- Insert additional steps into the flow.
- Copy the loadDocument and queryDocument services into another flow.
To create a load and query service with the wizard

1. On the File menu, click New.
2. In the New dialog box, select Flow Service, and click Next.
3. In the New Flow Service dialog box, next to Folder, select the folder in which you want to save the service.
4. In the Name field, type a name for the service, and click Next.
5. Select Load and Query an HTML Page and click Next. (Even though the option refers to an HTML document, you can use this option with XML documents, too.)
6. In the URL field, type the URL of the resource you want to query and click Next. (The URL you specify must begin with http: or https.)

Example http://www.rubicon.com/orders/orders.html

7. Do one of the following, depending on whether you are creating this service for an HTML or XML document:

   — If you are querying an HTML document and you want Developer to generate a set of default variables for an HTML document, select one or more of the following options:

   Select...  To create...
   Title      A String containing the title of the document.
   Anchors   A String List containing all the HREF= values from the document.
   Paragraphs A String List containing the text of every paragraph in the document.
   Tables    A String List containing the text from every table in the document.
   Forms     A Record List containing information about each form in the document.
   Lists     A String List containing the text from all of the list items in the document.

   — If you are querying an XML document, clear the settings in the New Flow Service dialog box. The default variables generated by the wizard are not useful for XML documents.
Do one of the following depending on whether you want Developer to automatically test (execute) the service it builds for you.

- If you want Developer to execute the service immediately after it builds it, select one of the following options and then click Finish.

<table>
<thead>
<tr>
<th>Select...</th>
<th>If you want Developer to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>Run the service from Developer and display the results in the Results tab.</td>
</tr>
<tr>
<td>Run in Browser</td>
<td>Run the service from your Internet browser (results will appear in the browser, not the Results tab).</td>
</tr>
<tr>
<td>Trace</td>
<td>Run the service in “trace” mode, which will display the parsed document on the Variables tab so you can use it to develop additional queries.</td>
</tr>
<tr>
<td>Step</td>
<td>Run the service in “step” mode. This will execute the loadDocument service and then stop.</td>
</tr>
</tbody>
</table>

- If you do not want Developer to execute the service immediately after it builds it, select No and then click Finish.

**Important!** If your service accesses a protected resource, it will not execute successfully using any of the execution options in the preceding step. This is because the loadDocument service does not have the auth values it needs to retrieve the requested document successfully. To create a load and query for a protected resource, select No in the preceding step. Then, after Developer generates the service, assign the appropriate username and password to the auth variable and execute the service manually from the Test menu.

**Using the Browser Recorder**

The wizard’s “browser recorder” allows you to generate a series of load and query steps by “recording” actions that you take in your Internet browser. When you use this method, the wizard monitors your browser session and automatically generates a load and query sequence for each hypertext link you select or HTML form you submit.

For example, if you used the Browser Recorder to capture the following actions, it would create a load and query sequence for each action as follows:

<table>
<thead>
<tr>
<th>If you...</th>
<th>The wizard will...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access a Web site</td>
<td>Create a load and query sequence for that URL.</td>
</tr>
<tr>
<td>Select a link from a Web site</td>
<td>Create a load and query sequence for that URL.</td>
</tr>
</tbody>
</table>
The Browser Recorder is useful for generating flows for complex Web sites and for capturing the URLs of documents that are accessed through Javascript or derived from an image map.

To generate a load and query service with the Browser Recorder

1. On the File menu, click New.
2. In the New dialog box, select Flow Service and click Next.
3. In the New Flow Service dialog box, next to Folder, select the folder into which you want to save this service.
4. In the Name field, type a name for the service, and click Next.
5. Select Record in Browser and click Next.
6. In the URL field, type the URL of the HTML page you want to query and click Record. (The URL you specify must begin with http: or https.)

   The wizard will launch your Internet browser and display the requested page. From this point forward, the wizard will capture the URLs of each page you visit via a hypertext link or an HTML form. The URLs of pages you visit are shown on the URLs Visited list.

   **Note:** You may want to arrange the windows on your screen so that you can see the wizard and your browser simultaneously.

   **Important!** The Browser Recorder cannot capture URLs that you type directly on the address line in your browser. If you want the Browser Recorder to record the address of a page that you visit, you must access that page through a hypertext link or an HTML form.

7. When you are finished recording, click Next.
8. If you want Developer to generate a set of default variables for an HTML document, select one or more of the following options, and then click Next.
Do one of the following, depending on whether you want Developer to automatically test (i.e., execute) the service it builds for you.

— If you want Developer to execute the service immediately after it builds it, select one of the following options and then click Finish.

<table>
<thead>
<tr>
<th>Select...</th>
<th>If you want Developer to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>Run the service from Developer and display the results in the Results tab.</td>
</tr>
<tr>
<td>Run in Browser</td>
<td>Run the service from your Internet browser (results will appear in the browser, not the Results tab).</td>
</tr>
<tr>
<td>Trace</td>
<td>Run the service in “trace” mode, which displays the parsed document on the Variables tab so you can use it to develop additional queries.</td>
</tr>
<tr>
<td>Step</td>
<td>Run the service in “step” mode, which causes Developer to execute the first service in the flow (loadDocument) and then halt. To run the next service (queryDocument), you must select that step in the Flow Editor and then select the Step command from the Test menu.</td>
</tr>
</tbody>
</table>

— If you do not want Developer to execute the service immediately after it builds it, select No and then click Finish.
Accessing Databases with Services

- The Types of Services You Can Create ........................................ 434
- Creating Database Flow Services ............................................. 435
- Creating Database Services with Java, C/C++, or VB .................. 442
- Creating Clients that Access Databases ..................................... 444
The Types of Services You Can Create

All types of services can access databases—flows, Java, C/C++, and Visual Basic. You can create:

- **Flow services using the Server Administrator.** When you create a flow service, you make selections from screens in the Server Administrator to describe the function that you want the service to perform. The server generates the flow service for you.

- **Java, C/C++, and Visual Basic services using SAP BC Server or your own development environment.** SAP BC Server provides built-in services that perform basic database operations, such as connecting to a database, selecting rows, inserting rows, and deleting rows. Your services can access these built-in services to perform database operations or use other database APIs (for example, JDBC) to access databases.

Deciding What Type of Service to Create

Use the information in the following table to decide which type of service you should create.

<table>
<thead>
<tr>
<th>Type of Service</th>
<th>Advantages and Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Flow services created with the Server Administrator</strong></td>
<td><strong>Advantages</strong>&lt;br&gt;You can easily create the flow service using the Server Administrator user interface.&lt;br&gt;You do not need to recompile code when you make changes to the flow service. <strong>Disadvantages</strong>&lt;br&gt;The flow service might not be able to handle nonstandard database features, such as Windows SQL types.&lt;br&gt;You can only make JDBC calls in a flow service. Use a Java, C/C++, or Visual Basic service to connect to a database using other libraries.</td>
</tr>
</tbody>
</table>
Creating Database Flow Services

Before you can use the Server Administrator to build flow services, the server must have a configured database alias for the database that you want to access. The database alias must be configured when you create a service and when the server executes the service. For more information about how to configure a database alias, see the SAP BC Administration Guide.

To create flow services using the Server Administrator, you go through a series of screens, selecting options and specifying information to indicate the function you want the service to perform. The server builds a flow service from the selections you specify.

The flow service invokes the same built-in services that you can invoke from Java, C/C++, and Visual Basic. For descriptions of the services, see “Built-in Database Services” on page 442.

You can choose from two options for how to create a flow service using the Server Administrator:

- **Generating from tables.** You select the table that you want the service to access. Then, you select the database operation you want your service to perform (Select, Insert, Delete, or Update). The server displays the columns in the table that you can use to indicate the input that the service expects. When you use this option, the server automatically generates the required SQL statements. For more information, see “Generating a Service from a Table” on page 439.

## Advantages and Disadvantages

<table>
<thead>
<tr>
<th>Type of Service</th>
<th>Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java, C/C++, or Visual Basic</td>
<td>- The service can be coded more flexibly in terms of processing data. You can manipulate data directly, rather than invoking SAP BC Server libraries. For example, you can use configuration files to guide processing settings.</td>
</tr>
<tr>
<td></td>
<td>- The service can include more robust error handling. For example, you can log errors to your own files or send e-mail notes to several people.</td>
</tr>
<tr>
<td></td>
<td>- The service can make direct calls to database APIs (for example, JDBC, ADO, and ODBC) to handle nonstandard database features.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- You must code and compile the service.</td>
</tr>
</tbody>
</table>

---

**Creating Database Flow Services**

Before you can use the Server Administrator to build flow services, the server must have a configured database alias for the database that you want to access. The database alias must be configured when you create a service and when the server executes the service. For more information about how to configure a database alias, see the SAP BC Administration Guide.

To create flow services using the Server Administrator, you go through a series of screens, selecting options and specifying information to indicate the function you want the service to perform. The server builds a flow service from the selections you specify.

The flow service invokes the same built-in services that you can invoke from Java, C/C++, and Visual Basic. For descriptions of the services, see “Built-in Database Services” on page 442.

You can choose from two options for how to create a flow service using the Server Administrator:

- **Generating from tables.** You select the table that you want the service to access. Then, you select the database operation you want your service to perform (Select, Insert, Delete, or Update). The server displays the columns in the table that you can use to indicate the input that the service expects. When you use this option, the server automatically generates the required SQL statements. For more information, see “Generating a Service from a Table” on page 439.
Generating from a SQL statement. You specify the SQL statement that you want the service to perform. You can specify more complex SQL statements than the server generates when you generate from tables. In addition, you can specify database-specific SQL. For example, use this option if you want to use database-specific SQL that requires the fully qualified table names. For more information, see “Generating a Service from a SQL Statement” on page 436.

After a service is created, you can update it using Developer in the same way you edit any other flow service. You can also invoke the flow service from other services.

Note: Regardless of how you generate them, database flow services are all standard flow services that call the pub.db:execSQL service. To examine a flow you have generated, open it in the Developer.

If you want to change the SQL statement the flow service executes, open that service in Developer and change the $dbsql parameter in the INVOKE execSQL step in the flow.

Generating a Service from a SQL Statement

When you specify that you want to generate the service from a SQL statement, you must select the SQL statement that you want the service to perform. You can specify a statement that is static or include either question marks (?) or template tags in the SQL statement to make it dynamic.

Important! The generated flow service is unable to properly call a stored procedure. It is unable to return the result set or output parameters. If you want to call a stored procedure, create the service in Java, C/C++, or Visual Basic and call the stored procedure using the pub.db:call service.

Specifying a Dynamic SQL Statement

When you specify a dynamic SQL statement, the service expects input values to replace the question marks or template tags that you specify. You can specify the following to make a SQL statement dynamic:

- **Question marks (?)**. Use a question mark in place of a single parameter that the service expects as input. When you use the Server Administrator to test the service, it recognizes the question mark and prompts for the required input.

  Example

  To select all rows from the Names table that match the values specified for the Last_Name column, specify the following SQL statement:

  ```sql
  select * from Names where Last_Name=?
  ```

  The service expects input for the value to use to match rows in the Last_Name column.
Example

To add a new row to the Addresses table and populate it with specified values, specify the following SQL statement:

```sql
insert into Addresses (name,street,city,state,zip) values (?,?, ?,?,?)
```

The service expects input for the values to use to populate the new row.

- **Template tags.** The template tags you can use in an SQL statement are the same tags you use in an output template. Besides allowing you to specify the values of individual input parameters, template tags also allow you to dynamically construct entire portions of the SQL statement at run time. (For a complete list of template tags, see Building Output Templates and DSPs.)

When you use template tags in an SQL statement, it cannot be tested with the Server Administrator. (The Server Administrator does not recognize template tags and will not prompt you for input when you execute the service) To test a service that uses template tags, you must open that service in Developer and add to its input parameters any variables that are referenced in a tag. Once you do this, you can test the service with Developer and it will prompt you for each input variable you defined. (For information about declaring input parameters for a service, see “Specifying Input Parameters” on page 82.)

Example

The following shows an SQL statement that selects all rows satisfying the criteria that will be specified in a variable named condition at run time:

```sql
select * from Names where %value condition%
```

At run time, the server will substitute the value of condition for the %value% tag. The following shows examples of what you might use as the value of condition:

```sql
name = 'Steve'
name like 'Jim'
```

Example

To delete all rows from the Music table that meet a specified condition, specify the following SQL statement:

```sql
delete from Music where %value outdated%
```

The service expects input for an input variable named outdated that it uses to replace the entire token. Following is an example of what a user can specify for %value outdated%:

```sql
ReleaseDate < '1950'
Format = '8-track'
```
Steps for Generating the Flow Service

Use the following procedure to generate a flow service by specifying a SQL statement.

**To generate a flow service from a SQL statement**

1. Open the Server Administrator if it is not already open. See the *SAP BC Administration Guide* if you need procedures for this step.
2. Select the Database task in the navigation area.
3. Click the Service Generation tab if it is not already displayed.
4. Select the database you want the service to access from the Source Alias list.
5. Select the package in which you want the service to reside from the list in the Package field.
6. Type the folder name for the service in the Folder field.
7. Type the name for the service in the Service field.
8. Click Generate from SQL.
9. Enter the SQL statement you want the service to execute in the Enter SQL statement section of the screen.
10. If you included template tags in the SQL statement, click the Process SAP BC template tags (%value, %ifvar, etc.) in this SQL statement check box.
11. Click Evaluate. The server displays the Input Binding Generation screen.
   - If you specified question marks (?) in the SQL statement for input values, the server displays a Bind parameters section of the screen. There is one Parameter n field for each question mark you specified in the SQL statement. The first question mark you specified corresponds to Parameter 0, the second to Parameter 1, and so forth.
12. Use this section of the screen to indicate additional information about the input parameters that a user will supply in place of the question marks. For each Parameter n field in the Bind parameters set the associated parameters as follows:

   - **For this parameter**
   - **Specify...**
   - **name:** The name that you want the service to use for the input value.
   - **type:** The data type of the input value. Select the correct data type from the list.
13. Click Generate.
Generating a Service from a Table

When you specify that you want to generate the service from database tables, the server displays information about the tables in the selected database. You are given the opportunity to narrow down (or restrict) the list of tables that the server displays. If your database contains many tables, narrow down the list of tables.

Restricting the List of Database Tables

You can restrict the list of database tables that the server displays by specifying one or more of the following:

- **Catalog.** You specify the name of the catalog whose tables you want to use.
  - If you are not working with a distributed database, do not restrict by catalog.
  - If you are working with a distributed database, you can specify the name of the database with which you want to work. If you are using DB2, use this field to specify the name of a DB2 location.

- **Schema pattern.** You specify the schemas whose tables you want to work with (if you are using DB2, use this field to specify an AuthID).
  - If you want tables for all schemas in the selected database, do not restrict by schema pattern.
  - If you want to restrict your search to tables by selected schemas, specify the schema name. You can specify a pattern-matching string if your JDBC supports it. Most drivers support the pattern-matching characters described for **Table name pattern** below. However, check your driver’s documentation for information about its pattern-matching capabilities.

- **Table name pattern.** You specify the names of the tables with which you want to work. Specify a table name or a pattern-matching string that specifies the names of the tables. Most drivers support the following pattern-matching characters; however, check your driver’s documentation for information about its pattern-matching capabilities.

<table>
<thead>
<tr>
<th>Use</th>
<th>To match</th>
<th>For example</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>Any string of characters</td>
<td>HR% matches: HR30all, HR15mgmt, HR01payroll, and so forth.</td>
</tr>
<tr>
<td>_</td>
<td>A single character</td>
<td>HR3_mgmt matches: HR30mgmt, HR31mgmt, HR3Amgmt, and so forth.</td>
</tr>
</tbody>
</table>

- **Table type.** The type of the tables for which you want information. You can choose from the following common JDBC table types: Table, View, System Table, Global Temporary, Local Temporary, Alias, and Synonym.
Steps for Generating the Flow Service

Use the following procedure to generate a flow service by specifying a SQL statement.

To generate a flow service from a table

1. Open the Server Administrator if it is not already open. See the SAP BC Administration Guide if you need help with this step.
2. Select the Database task in the navigation area.
3. Click the Service Generation tab if it is not already displayed.
4. Select the database you want the service to access from the Source Alias list.
5. Select the package in which you want the service to reside from the Package field.
6. Type the folder name for the service in the Folder field.
7. Type the name for the service in the Service field.
8. Click Generate from table. If you are accessing a large database, you might want to restrict your search to the tables with which you want to work.
   a. To restrict your search, fill in one or more fields in the Restrictions section of the screen, and then click Connect. For more information about how to restrict the search, see “Restricting the List of Database Tables” on page 439.
   b. To search all database tables, click Connect.
9. In the Select a table section of the screen, click the table name that you want to use to generate the service.
10. Select the database operation that you want the service to perform (Select, Insert, Delete, or Update) from the Service type section of the screen.
11. Select the columns that you want to use as input in the Columns section of the screen. When the service is executed, it will expect input values for each of the columns you select. The service uses the input values as follows:

For this operation... The service...
Select          Selects rows that have columns that match the input values.
Insert          Populates the columns of the inserted rows using the input values. If you want to populate all columns in the row, select all column names.
Creating Database Flow Services

For this operation... The service...
Delete Selects rows that have the columns that match the input values, and deletes the selected rows.
Update Selects rows and updates them.

You identify the criteria to use to select rows by checking column names in the Criteria column. The service expects input values for each column name you check. The service selects rows that have columns that match the input values. If you want to select all rows, do not check any Criteria columns.

To identify the columns to update in the selected rows, use the Set column. The service expects input values for each column you check. The service sets the value of the checked columns (for all selected rows) to the specified input values.

12 Click Generate SQL.

If you selected any columns for input variables, the server displays a Bind parameters section of the screen. There is one Parameter n field for each input variable you specified in the SQL statement.

13 Use this section of the screen to indicate additional information about the input parameters. For each Parameter n field in the Bind parameters set the associated parameters as follows:

For this parameter... Specify...
name: The name that you want the service to use for the input value. By default the server uses the database column name. If you want to use a different name, type the new name in the name: field.
type: The data type of the input value. By default, the server uses the data type that is associated with the database column. If you want to use a different data type, select the data type from the list.

14 Click Generate.
Output from the Flow Service

The following describes the output from a flow service that you generate from a database table based on the database operation that the service performs:

<table>
<thead>
<tr>
<th>Database Operation</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>Number of rows that the service retrieved and the columns from those rows that the service requested.</td>
</tr>
<tr>
<td>Insert</td>
<td>Update count that the database returned in response to the SQL command.</td>
</tr>
<tr>
<td>Delete</td>
<td>Update count that the database returned in response to the SQL command.</td>
</tr>
<tr>
<td>Update</td>
<td>Update count that the database returned in response to the SQL command.</td>
</tr>
</tbody>
</table>

Creating Database Services with Java, C/C++, or VB

You can code services that access databases in Java, C/C++, or Visual Basic. Code your own services if you need a service that performs more complex database operations than a flow service can provide.

To assist you in coding database services, you can use the built-in database services to perform basic database operations and test SQL statements before you add them to your services.

If you want your service to access a database that has nonstandard features (for example, data types that are not supported by SQL), use other database connection APIs. For example, you can make direct calls to JDBC or use other connection libraries, such as ADO.

Built-in Database Services

SAP BC Server provides several built-in database services that perform basic database operations. These services use JDBC to connect to the database to perform the specific database operation. You can invoke the built-in database services from:

- Java, C/C++, and Visual Basic services
- Any flow service that you create in Developer by using the INVOKE flow step
- Clients

The *SAP BC Built-In Services Guide* contains a list of built-in database services and shows input and output information for each. Your service or client must invoke the built-in
database service that opens a connection to a database before it can invoke any of the other built-in database services.

Testing SQL Statements

You can use the Server Administrator to test an SQL statement that you want to execute using the built-in database service, `pub.db:execSQL`, or that you want to invoke directly using a JDBC call.

Before you can use the Server Administrator to test an SQL statement, the server must have a configured database alias for the database that you want to access. For more information about how to configure a database alias, see the SAP BC Administration Guide.

To test a SQL statement

1. Open the Server Administrator if it is not already open. See the SAP BC Administration Guide if you need procedures for this step.
2. Select the Database task in the navigation area.
3. Click the Service Generation tab if it is not already displayed.
4. Select the database for which you want to issue a SQL statement from the Source Alias list.
5. Select any package from the Package list, and type a folder name in the Folder field and a service name in the Service field. It does not matter what you specify in these fields because you will not be creating a service.
6. Click Generate from table.
7. If you are accessing a large database, you might want to restrict your search to the tables with which you want to work.
   - To restrict your search, fill in one or more fields in the Restrictions section of the screen, and then click Connect. For more information about how to restrict the search, see “Restricting the List of Database Tables” on page 439.
   - To search all database tables, click Connect.
8. In the Test SQL Queries section of the screen, type the SQL statement you want to test.
9. Click Execute query.

The server displays a screen that lists the results from your SQL query.
Error Handling

Some database services return a $dbMessage output value which contains a text message that describes the results of the service. If the service results in an error, the service also returns standard error output values.

Creating Clients that Access Databases

You can access the databases from all types of clients:
- Browser-based clients
- Clients coded in Java, C/C++, or Visual Basic

Invoking a Database Service from a Browser-based Client

Several of the database services can accept input from a browser-based client. (The descriptions of the database services in the SAP BC Built-In Services Guide indicate whether you can invoke a built-in database service from a browser-based client.) When a browser-based client submits input variables containing row information, these services automatically convert the row information into a nested IData object. The service determines which variables to convert based on the names of the variables. All variables except those whose names begin with _ or $db are converted into nested IData objects.

For example, if a browser user submits the following name-value pairs:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$dbTable</td>
<td>Table Name</td>
</tr>
<tr>
<td>Name</td>
<td>Joe B</td>
</tr>
<tr>
<td>Company</td>
<td>Widgets, Inc.</td>
</tr>
</tbody>
</table>

The pub.db:insert service converts the inputs as follows:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$dbTable</td>
<td>Table Name</td>
</tr>
<tr>
<td>$data</td>
<td>Variable Name</td>
</tr>
<tr>
<td></td>
<td>Contents</td>
</tr>
<tr>
<td>Name</td>
<td>Joe B</td>
</tr>
<tr>
<td>Company</td>
<td>Widgets, Inc.</td>
</tr>
</tbody>
</table>
Invoking a Built-in Service from a Java, C/C++, or VB Client

You can create client applications in Java, C/C++, or Visual Basic. The client applications can use the built-in database services to perform database operations. For information about these built-in services, see the SAP BC Built-In Services Guide.

If you want your client to access a database that has nonstandard features (for example, data types that are not supported by SQL), you use other database APIs. For example, you can make direct calls to JDBC or use other connection libraries, such as ADO.

Sample Code - IData

The following shows a sample Java client that accesses a database using an IData object to pass and receive data from the database service.

```
import com.wm.data.*;
import com.wm.app.b2b.client.Context;

public class DBClient {
    public static void main (String [] args) throws Exception {
        IData in = null;
        IData out = null;
        IData criteria = null;
        IData set = null;

        // connect to your integration server using an appropriate user
        // name and password (doesn't have to be Administrator)
        Context ctx = new Context();
        ctx.connect ("localhost:5555", "Administrator", "manage");

        // (1) request a DB connection by DB alias (if the DB
        // changes location or something, we won't have to change
        // this client code)
        in = IDataFactory.create();
        IDataCursor inCursor = in.getCursor();
        inCursor.insertAfter ("$dbAlias", "Employees");
        inCursor.destroy();
        out = ctx.invoke ("wm.util.db", "connect", in);

        // (2) update the Identification table to set Fonz's ID
```
// to 6500. note that we couldn't do this from a Web // browser because we couldn't build up the complex // nested data structures //
in = IDataFactory.create();
IDataCursor inCursor = in.getCursor();
inCursor.insertAfter("$dbAlias", "Employees");
inCursor.insertAfter("$dbTable", "Identification");
criteria = IDataFactory.create();
IDataCursor criteriaCursor = criteria.getCursor();
criteriaCursor.insertAfter("name", "fonzie");
criteriaCursor.destroy();
inCursor.insertAfter("$criteria", criteria);
set = IDataFactory.create();
IDataCursor setCursor = set.getCursor();
setCursor.insertAfter("ID", "6500");
setCursor.destroy();
inCursor.insertAfter("$set", set);
inCursor.destroy();
out = ctx.invoke("wm.util.db", "update", in);

//
// (3) look at the return values (updateCount is the // most important in this case)
//
IDataCursor outCursor = out.getCursor();
try {
    if (outCursor.first("updateCount")) {
        int uc = Integer.parseInt((String)outCursor.getValue());
        System.err.println("Update count: "+uc);
    } else  
        System.err.println("Error: no update count returned");
    outCursor.destroy();
} catch (Exception e) {
    // maybe something went wrong with the DB access; we // can get more information here
    outCursor.first("$error");
    System.err.println("Error: "+outCursor.getValue());
    outCursor.first("$errorType");
    System.err.println("Error type: "+outCursor.getValue());
    outCursor.destroy();
}
}
Sample Code - Values

The following shows a sample Java client that accesses a database using a Values object to pass and receive data from the database service. (Note that Values has been replaced by IData object. For new code development, we recommend that you use IData, instead.)

```java
import com.wm.util.Values;
import com.wm.app.b2b.client.Context;

public class DBClient {
    public static void main (String [] args) throws Exception {
        Values in = null, out = null;
        Values criteria = null, set = null;

        // connect to your integration server using an appropriate user
        // name and password (doesn't have to be Administrator)
        //
        Context ctx = new Context();
        ctx.connect ("localhost:5555", "Administrator", "manage");

        // (1) request a DB connection by DB alias (if the DB
        // changes location or something, we won't have to change
        // this client code)
        //
        in = new Values();
        in.put ("$dbAlias", "Employees");
        out = ctx.invoke ("wm.util.db", "connect", in);

        // (2) update the Identification table to set Fonzie's ID
        // to 6500. note that we couldn't do this from a Web
        // browser because we couldn't build up the complex
        // nested data structures
        //
        in = new Values();
        in.put ("$dbAlias", "Employees");
        in.put ("$dbTable", "Identification");
        criteria = new Values();
        criteria.put ("name", "fonzie");
        in.put ("$criteria", criteria);
        set = new Values();
        set.put ("ID", "6500");
        in.put ("$set", set);
        out = ctx.invoke ("wm.util.db", "update", in);

        // (3) look at the return values (updateCount is the
```
try {
    int uc = Integer.parseInt ((String)out.get("updateCount"));
    System.err.println ("Update count: "+uc);
} catch (Exception e) {
    // maybe something went wrong with the DB access; we
    // can get more information here
    System.err.println ("Error: "+out.get("$error");
    System.err.println ("Error type: "+out.get("$errorType");
}
CHAPTER 18

Using Guaranteed Delivery

- What Is Guaranteed Delivery? .................................................. 450
- Indicating You Want to Use Guaranteed Delivery ....................... 450
- How Transactions Are Managed ............................................. 450
- Identifying Transactions ....................................................... 450
- Specifying How Long Transactions are Active ......................... 452
- Handling Failures ............................................................... 453
- Creating a Java Client that Uses Guaranteed Delivery ................. 454
- Creating a Flow Service that Uses Guaranteed Delivery ............. 458
What Is Guaranteed Delivery?

Guaranteed delivery is a facility of SAP BC Server that ensures guaranteed, one-time execution of services. It protects transactional requests from transient failures that might occur on the network, in the client, or on the server.

A transient failure is a failure that can correct itself within a specified period of time. If a request cannot be delivered to the server due to a transient failure, the request is resubmitted. If the problem corrected itself, the request is successfully delivered on a subsequent attempt. You can determine what constitutes a transient error by specifying a time-to-live (TTL) period for a guaranteed delivery transaction and, optionally, the number of times a transaction should be retried. If you do not specify the TTL or retry value, the configured defaults are used.

You can use guaranteed delivery when you invoke a service from a client or from within another service.

Important! You can only use the guaranteed delivery capabilities with stateless (i.e., atomic) transactions. As a result, guaranteed delivery capabilities cannot be used with multi-request conversational services.

Indicating You Want to Use Guaranteed Delivery

To invoke services using guaranteed delivery from either a client application or another service use the class watt.client.TContext (TContext) that is part of the Client API. Similar to the standard class watt.client.Context (Context), you use TContext to request that SAP BC Server execute a service. However, the server performs guaranteed delivery functions when a client application or service requests services through TContext.

How Transactions Are Managed

Guaranteed delivery transactions are managed by Job Managers. For client applications, the Job Manager runs on the client. For services, the Job Manager runs on the server.

The Job Managers manage all guaranteed delivery transactions that a process creates using TContext. The Job Managers maintain a job store of the guaranteed delivery transactions. The job store contains a record for each transaction. In addition, the Job Managers maintain a log that tracks the progress of all transaction operations.

The Job Manager handles the invocation of the service using background threads, which the Job Manager allocates from a configurable pool of threads. The Job Manager sends the service requests to a SAP BC Server and accepts the results on behalf of the client applications or services that use TContext. If the Job Manager does not receive a result for a transaction in its job store, it resubmits that request to execute the service. It continues to resubmit requests until it either receives a result or the transaction expires.
Customizing the Job Manager

You can customize how the Job Manager manages guaranteed delivery transactions programmatically or through system properties. To specify programmatically, your client application must specify the setting with the parameters of TContext methods. To specify through system parameters, specify the setting on the Java command line.

If a setting is specified both with a parameter of TContext and through a system property, the Job Manager uses the setting specified through the system property.

- **Location of the Job Store.** Specify the directory in which the Job Manager maintains its job store.

  - **TContext Method:**
    - Client apps: Specify using a parameter with the init method.
    - Services: Cannot specify using a TContext method.
  - **System Property:** Use the `-Dwatt.tx.jobdir=directory` option.

  The default for a client application is: `./jobs`
  The default for a service is: `<sapbc>/server/logs/jobsout`

- **Location of the transaction Log.** Specify the file in which the Job Manager maintains its audit-trail log of all the guaranteed delivery transaction operations it processes.

  - **TContext Method:**
    - Client apps: Specify using a parameter with the init method.
    - Services: Cannot specify using a TContext method.
  - **System Property:** Use the `-Dwatt.tx.logfile=filename` option

  The default for a client application is: `./txout.log`
  The default for a service is: `<sapbc>/server/logs/tx.log`

- **Submission interval for the Job Store.** Specify the number of seconds between sweeps of the job store. The Job Manager sweeps the job store to submit transactions to a SAP BC Server.

  - **TContext Method:** Cannot specify using a TContext method.
  - **System Property:** Use the `-Dwatt.tx.sweepTime=seconds` option.

  The default is: 60 seconds
CHAPTER 18 Using Guaranteed Delivery

- **Time to Retry Interval.** Specify the number of seconds to wait after a service request failure before the Job Manager resubmits the request to SAP BC Server.

  **TContext Method:** Cannot specify using a TContext method.
  
  **System Property:** Use the `-Dwatt.tx.retryBackoffTime=seconds` option.

  The default is: 60 seconds

- **Number of Client Threads in Thread Pool.** Specify the number of threads you want to make available in a thread pool to service pending requests.

  **TContext Method:** Cannot specify using a TContext method.
  
  **System Property:** Use the `-Dwatt.tx.jobThreads`.

  The default is: 5 threads

### Identifying Transactions

It is the responsibility of the client application or service to obtain a transaction ID (tid) for each guaranteed delivery request and to specify the transaction ID with each subsequent request for the transaction.

The client application or service obtains the transaction ID from SAP BC Server using the `startTx()` method, which is used to start a guaranteed delivery transaction. See “Creating a Java Client that Uses Guaranteed Delivery” on page 454 for additional instructions and sample code.

### Specifying How Long Transactions are Active

A guaranteed delivery transaction has two attributes that determine how long it stays active: the time-to-live (TTL) and the retry limit. The TTL specifies the number of minutes that a transaction is to remain active. The retry limit specifies the maximum number of times that the Job Manager is to resubmit a request. A transaction becomes inactive when the TTL or the retry limit (if specified) is reached, whichever comes first. When a transaction becomes inactive, it remains in the job store, but the Job Manager no longer attempts to submit the request.

The client application or service sets the TTL (and optionally, the retry limit) with the `startTx()` method, which it uses to start a guaranteed delivery transaction. See “Creating a Java Client that Uses Guaranteed Delivery” on page 454 for additional instructions and sample code.

These values determine the degree of tolerance the client application or service has towards transient network and server errors that occur at run time. Specifically, they determine the length of the outage that the client application or service considers...
Handling Failures

If a non-transient error prevents your client application or service from receiving the results from a service request, your application will receive an error message.

Records remain in the job store for a transaction until the client application or service explicitly ends the transaction. To avoid exhausting the job store, a client application or service must make sure to complete all the transactions it starts, or a site must establish administrative procedures to address failed jobs.

TContext can return the following types of errors:

- `AccessException`. The client application or service either supplied invalid credentials or is denied access to the requested service.
- `ServiceException`. The service encountered an execution error.
- `DeliveryException`. The Job Manager failed and became disabled. An administrator should be notified to correct this problem. For client applications, code your client application to notify an administrator when this type of error occurs. After the problem is corrected, re-enable the Job Manager using the `TContext.resetJobMgr()` method.

For services, guaranteed delivery notifies the administrator identified by the `watt.server.txMail` configuration setting. After the problem is corrected, re-enable the Job Manager by executing the `wm.server.tx.resetOutbound` service.

- `IllegalRequestException`. The client application or service made an invalid request; for example, supplied an invalid transaction ID (tid) or other invalid parameter.
- `TXException`. A failure occurred with the transaction. The transaction timed out, hit the retry limit, or encountered a heuristic error. Typically, this type of error indicates that the transaction became inactive either because the time-to-live (TTL) value elapsed or the retry limit was met. To distinguish between these two errors, use the `isExceededRetries()` method.

Heuristic errors will only occur if you altered the default configuration of SAP BC Server to fail PENDING requests when SAP BC Server is restarted after a failure. Use the `isHeuristicFailure()` method to determine if a heuristic error occurred.

Note: A heuristic error does not guarantee that your transaction was not executed, only that its results could not be returned. Keep this in mind if you are processing transactions that must be executed once and only once (for example, an application that enters purchase orders or pays invoices). You might also need to implement additional mechanisms in your client application or service to ensure that a transaction does not get posted twice.
Creating a Java Client that Uses Guaranteed Delivery

When using guaranteed delivery for a client application, you must initialize TContext when a process starts. The server handles this function when a service uses guaranteed delivery.

Create TContext instances for different connection attributes. If you are only connecting to one host with a single set of credentials, you need only one TContext regardless of how many threads share the TContext.

The main difference between Context (the standard class) and TContext is that your client application or service is responsible for obtaining a transaction ID (tid) and associating it with each request you make for the same transaction. You receive a transaction ID (tid) when you start a guaranteed delivery transaction.

After a transaction is started and a transaction ID is received, you can invoke a service using guaranteed delivery. You must supply the transaction ID when you invoke the service.

When the transaction completes, you must end the transaction to clear the record for the transaction from the Job Manager’s job store.

You can chain transactions in a sequence so that each transaction in a sequence waits until the preceding transaction executes. To chain transactions, supply the transaction ID (tid) from the previous transaction when starting a new transaction.

When you are done executing guaranteed delivery transactions for a specific instance, disconnect to end the instance of TContext. When you disconnect, TContext unregisters the instance with the Job Manager.

After a client application disconnects all TContext instances, it should shutdown guaranteed delivery for the process. The server handles this function when a service uses guaranteed delivery. If your client application or service has active TContext instances when the shutdown occurs, the server throws an exception (unless the shutdown was performed with the force option).

Sample Code (Synchronous Request)

The following code fragment illustrates the basic steps required to submit a synchronous request to the Job Manager. Synchronous requests are submitted using the invokeTx method. You can also submit asynchronous requests to the Job Manager as shown in the next section.

**Important!** To compile the following sample code (or any Java client that uses guaranteed delivery), you must include the following Import statements in your Java program.

```java
Import com.wm.app.b2b.client.*;
Import com.wm.util.*;
```
TContext tc = null;

// initialize TContext and establish connection attributes
try {
    TContext.init("./jobs", ".\tx.log");
tc = new TContext();
tc.connect("localhost:5555", null, null);
} catch (ServiceException e) {
    System.err.println("Error: "+e.getMessage);
    System.exit(-1);
}

// do work with TContext - get tid, call service, end tid
try {
    String tid = tc.startTx(3);
    Values result = tc.invokeTx(tid, "wm.server", "ping", new
    Values());
    System.out.println("Result="+result.toString());
tc.endTx(tid);
} catch (TXException e) {
    System.err.println("Job Failed: "+e.getMessage());
    System.exit(-1);
} catch (DeliveryException e) {
    System.err.println("JobMgr Disabled: "+e.getMessage());
    System.exit(-1);
} catch (AccessException e) {
    System.err.println("Access Denied: "+e.getMessage());
    System.exit(-1);
} catch (ServiceException e) {
    System.err.println("Error: "+e.getMessage());
    System.exit(-1);
}

// release connection and shutdown
try {
    tc.disconnect();
    TContext.shutdown();
} catch (ServiceException e) {
    System.err.println("Error: "+e.getMessage());
    System.exit(-1);
}

# Step Description
1 Declare TContext Declare TContext as a variable.
2 Initialize TContext. Initialize TContext and specify the job store directory and
audit-trail log. The Job Manager starts.

Important! Do not include this step if your client will run as
a service on a SAP BC Server. This function is
automatically performed by the server and must not be
included in your code.
# Step | Description
--- | ---
3 | Instantiate TContext. Create a new TContext object.  
4 | Establish connection attributes for the TContext instance. Execute connect() to specify the SAP BC Server on which you want to invoke services using this context.  
| **Note:** Multiple threads can share an instance of TContext as long as they use the same connection attributes—i.e., they use the same SAP BC Server and user ID/password established by that instance of TContext.  
4 | Establish connection attributes for the TContext instance. Execute connect() to specify the SAP BC Server on which you want to invoke services using this context.  
| **Note:** Multiple threads can share an instance of TContext as long as they use the same connection attributes—i.e., they use the same SAP BC Server and user ID/password established by that instance of TContext.  
5 | Start the transaction. Execute startTx() to obtain a transaction ID (tid) and specify the transaction time-to-live (TTL).  
6 | Invoke the service. Execute invokeTx() to invoke a service.  
| **Note:** This particular example passes input and receives output in a Values object. You could also use an IData object to do this.  
7 | End the transaction. Execute endTx() to end the transaction. This method clears the record for this transaction from the Job Manager's job store.  
8 | Check for errors. Check for the different types of errors. Always check for Service Exceptions last.  
9 | Close the session on SAP BC Server. Execute disconnect to end the use of this instance of TContext. The application should not perform this step until it is done because disconnect unregisters TContext with the Job Manager.  
10 | Shutdown The Job Manager ends.  
| **Important!** Do not include this step if your client will run as a service on a SAP BC Server. This function is automatically performed by the server and must not be included in your code.  

For additional information about TContext and its methods, see the TContext class in the [SAP BC Java API Reference](<sapbc>\developer\doc\api\index.html).
Creating a Java Client that Uses Guaranteed Delivery

Sample Code (Asynchronous Request)
:The following example illustrates the steps you take to submit an asynchronous request
to the Job Manager. To submit an asynchronous request, you establish a connection and
start a transaction just like you do for a synchronous request. However, you submit the
request using the submitTX method instead of the invokeTx method. Then, you must
retrieve the results of the request using the retrieveIDTx method (to get results as an IData
object) or the retrieveTx method (to get results as a Values object).
/**
* Sample of a Java TContext client that uses SSL to perform a GD transaction
*/
import com.wm.app.b2b.client.*;
import com.wm.util.*;
public class TCSample {
public static void main (String[] args)
{
TContext tc = null;
String privkey = "./config/privKey1.der";
String[] certFiles = {"./config/cert1.der","./config/cacert1.der"};
// initialize TContext and establish connection attributes
try {
TContext.init("./jobs", "./tx.log");
tc = new TContext();
tc.connect("localhost:5555", null, null);
tc.setSecure(true);
tc.setSSLCertificates(privKey,certFiles);

SAP BC Developer Guide 4.8



457


Creating a Flow Service that Uses Guaranteed Delivery

Using the services in the pub.remote.gd folder, you can build flow services that submit requests to other SAP BC Servers through guaranteed delivery.

The following examples show how you would submit both synchronous and asynchronous requests using the built-in services. For a description of these services, see the SAP BC Built-In Services Guide.

For additional information about TContext and its methods, see the TContext class in the SAP BC Java API Reference.
Sample Flow (Synchronous Request)

The following flow illustrates the basic steps you use to execute a synchronous transaction from a flow service.

Flow Service that executes a synchronous transaction

<table>
<thead>
<tr>
<th>Step</th>
<th>Invoke this Service...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>pub.remote.gd:start</td>
<td>Start the transaction. When you invoke this service, you specify the alias for the SAP BC Servers to which you want to submit a request as well as transaction-related parameters such as time-to-live and followid. This service returns a tid as output.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: Internally, this service opens a session on the server and performs startTx, so there is no need for you to explicitly open a session on the server like you must do in a Java guaranteed-delivery client.</td>
</tr>
<tr>
<td>2</td>
<td>pub.remote.gd:invoke</td>
<td>Invoke the service. You must provide the tid (produced by start, above), the name of the requested service, and the input values for that service as input. This service returns the results from the remote service as output.</td>
</tr>
<tr>
<td>3</td>
<td>pub.remote.gd:end</td>
<td>End the transaction. You must call this service to clear the transaction from the job store. It takes the tid as input.</td>
</tr>
</tbody>
</table>
Sample Flow (Asynchronous Request)

The following flow illustrates the basic steps you use to execute an asynchronous transaction from a flow service.

Flow Service that executes an asynchronous transaction

<table>
<thead>
<tr>
<th>Step</th>
<th>Invoke this Service...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>pub.remote.gd:start</td>
<td>Start the transaction. When you invoke this service, you specify the alias for the SAP BC Server to which you want to submit a request as well as transaction-related parameters such as time-to-live and followid. This service returns a tid as output.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: Internally, this service opens a session on the server and performs startTx, so there is no need for you to explicitly open a session on the server like you must do in a Java guaranteed-delivery client.</td>
</tr>
<tr>
<td>2</td>
<td>pub.remote.gd:submit</td>
<td>Submit the service request. You must provide the tid (produced by start, above), the name of the requested service, and the input values for that service as input.</td>
</tr>
<tr>
<td>3</td>
<td>pub.remote.gd:getStatus</td>
<td>Check for results. You can optionally use a REPEAT step to poll the job store and check whether the results from the transaction have been received. This service returns “DONE” when results are available.</td>
</tr>
<tr>
<td>4</td>
<td>pub.remote.gd.retrieve</td>
<td>Retrieve the results. This service returns the results from the service request you submitted earlier. It takes the tid as input.</td>
</tr>
<tr>
<td>Step</td>
<td>Invoke this Service...</td>
<td>To...</td>
</tr>
<tr>
<td>------</td>
<td>------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>5</td>
<td>pub.remote.gd:end</td>
<td><em>End the transaction.</em> You must call this service to clear the transaction from the job store. It takes the tid as input.</td>
</tr>
</tbody>
</table>
Working with MIME and S/MIME Messages

- Overview ................................................. 464
- What Is MIME? ........................................... 464
- What Is S/MIME? .......................................... 467
- The MIME and S/MIME Services .................. 472
- MIME Messages, MIME Entities and MIME Objects 473
- Building MIME and S/MIME Messages .......... 473
- Extracting Data from MIME and S/MIME Messages 492
Overview

SAP BC Server provides built-in services that let you construct MIME messages, secure them, and transport them over the Internet. It also provides services that let you extract information from MIME messages that are passed into the pipeline, decrypting them if necessary.

What Is MIME?

MIME — Multipurpose Internet Mail Extensions — is a standard yet flexible message format that is used to represent messages for transmission over the Internet. The MIME extensions were added to the Simple Mail Transport Protocol (SMTP) to allow e-mail transmissions to carry more than simple, 7-bit, textual messages.

The MIME standards allow for the transmission of:

- Non-textual content such as images, audio clips, and other binary files
- Messages in character sets other than US-ASCII
- Multiple files in a single transmission

Although originally developed for the SMTP protocol, MIME can be used by other Internet technologies (such as HTTP) as a standard messaging format.

Basic Structure of a MIME Message

Like a standard mail message, a MIME message has two basic components: a set of header fields and a body.

A Simple MIME Message

Date: Mon, 08 Jan 2001 10:35:23 -0500
From: "Exprint Estimating" <EXPEst@exprint.com>
To: "Purch01@GSX.com" Purch01@GSX.com
MIME-Version: 1.0

Dear Buyer,
EXP Printing has received your request for an estimate. Your will receive a formal quote within 24 hours. Your request number is RPQ-0A000011318. Questions? Contact customer service from 8:00am and 6:00pm ET at 800-334-2517.
Header Fields

Header fields provide information about the structure and encoding of a message. They consist of name:value pairs and appear at the top of the message. A MIME-compliant message must contain the “MIME-Version” header field.

Besides the MIME-Version header field, most messages have additional fields that supply information to the agent, transport, or application that will convey or consume the message. For example, when a MIME message carries anything other than plain, US-ASCII text, it must include the “Content-Type” header field. Messages that are routed over SMTP will also have the “Date,” “To,” and “From” header fields.

A message may also contain custom header fields that are specific to a particular agent or application. Such application-specific header fields must be prefixed with the characters “X-” to distinguish them from the standard header fields defined by the MIME and/or transport protocols.

This chapter does not attempt to describe the purpose or use of individual header fields. However, to use MIME effectively, you will need to understand which header fields your solution requires and know how to set them or interpret them correctly. For information about header fields, see the following references.

<table>
<thead>
<tr>
<th>Reference</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC 2076 – Common Internet Message Headers</td>
<td><a href="http://www.imc.org/rfc2076">http://www.imc.org/rfc2076</a></td>
</tr>
<tr>
<td>RFC 822 – Standard Format of Internet Text messages</td>
<td><a href="http://www.imc.org/rfc822">http://www.imc.org/rfc822</a></td>
</tr>
<tr>
<td>RFC 2045 – Multipurpose Internet Mail Extensions</td>
<td><a href="http://www.imc.org/rfc2045">http://www.imc.org/rfc2045</a></td>
</tr>
<tr>
<td>RFC 2046 – MIME Media Types</td>
<td><a href="http://www.imc.org/rfc2046">http://www.imc.org/rfc2046</a></td>
</tr>
<tr>
<td>RFC 2047 – MIME Header Extensions for Non-ASCII</td>
<td><a href="http://www.imc.org/rfc2047">http://www.imc.org/rfc2047</a></td>
</tr>
<tr>
<td>RFC 2048 – MIME Registration Procedures</td>
<td><a href="http://www.imc.org/rfc2048">http://www.imc.org/rfc2048</a></td>
</tr>
<tr>
<td>RFC 2049 – MIME Conformance Criteria</td>
<td><a href="http://www.imc.org/rfc2049">http://www.imc.org/rfc2049</a></td>
</tr>
</tbody>
</table>

The Body

The body of a MIME message contains the actual content of the message. It is separated from the last header field by a blank line—a two-byte sequence consisting of an ASCII carriage return (CR) and an ASCII linefeed (LF) on a line by itself.

The message body can contain any sequence of data, including additional MIME messages. It is sometimes referred to as the payload. When you send an e-mail message, the body of your letter resides in the body of a MIME message. Similarly, when you attach a file to an e-mail message, the content of the file is carried in the body of a MIME message.
CHAPTER 19 Working with MIME and S/MIME Messages

Multipart MIME Messages

One of the key reasons for the development of MIME was to allow the transmission of multiple files (payloads) in a single message. When a MIME message contains multiple payloads, it has two kinds of header fields: message headers, which appear only at the beginning of the message, and part headers, which appear at the beginning of each body part.

Message headers apply to the entire message. Part headers apply only to the body part in which they appear. The following example shows a MIME message with two body parts.

A Multipart MIME Message

If a MIME message has more than one payload, its Content-Type header field must be set to a multipart content type (e.g., Content-Type:multipart/mixed or Content-Type:multipart/alternative), and it must declare a boundary-separator. The
boundary separator is a string that delimits body parts. It must appear before and after each part in the message. (In the example above, the string X----B164240404-----X is the boundary separator.)

Note: You may have noticed that the string separating the body parts in the preceding example includes a few extra characters that are not part of the separator string declared in the Content-Type header field. This is because the MIME format requires that two dash characters precede each separator in the message, and two dash characters follow the last separator string in the message.

When you build a multipart message with SAP BC Server, it automatically sets the Content-Type header to “multipart,” declares the separator string, and inserts the boundary separators for you.

What Is S/MIME?

S/MIME—Secure Multipurpose Internet Mail Extensions—is a standard message format that allows MIME messages to be exchanged securely between parties over the Internet. It provides two security mechanisms—digital signatures and encryption—based on RSA technology and the Public Key Infrastructure (PKI).

Digital Certificates

PKI employs a system of credentials known as digital certificates—electronic documents that represent and identify individual users. A digital certificate is like an electronic identification card. It positively identifies a particular individual, organization, or application.

Besides providing information about the owner of the certificate (name, organization, e-mail address, and so forth), a digital certificate holds the owner’s public key. Under public/private key technology, a certificate owner has two keys. Parties that want to exchange messages securely with the certificate owner, use the public key published on the owner’s certificate. Transmissions secured with a public key can only be successfully processed with the corresponding private key—a secret key that only the certificate owner has.

Digital certificates are issued and signed by Certificate Authorities (CAs). A CA is similar to a notary public. Its signature vouches for the identity of the individual or organization named on the certificate and attests to the validity of the public key. It also “seals” the certificate with a digital signature, which certifies the certificate’s contents and prevents it from ever being altered undetected. VeriSign® and Entrust® are examples of public CAs. They are considered “root-level” entities. Other intermediaries, such as financial institutions, are also permitted to issue certificates under the authority of a root CA.

You cannot verify the authenticity of a certificate without having the certificate of the CA that issued it. If the issuing CA is an intermediary, you must also have the certificate of its
CA. The set of certificates required to trace the authenticity of a certificate back to a trusted CA is called a certificate chain.

**Note:** To authenticate a certificate, some recipients require a complete certificate chain—one that extends all the way back to a root-level CA—while others are satisfied with a partial chain that goes back to a specific intermediary. Always submit a complete chain unless you know for certain that the recipient accepts partial chains.

**Digital Signatures**

A digital signature is a special block of data affixed to a message that assures the identity of the sender and the integrity of the message.

A digital signature secures a message in several ways. First, it contains the sender’s digital certificate. This allows a recipient to identify the sender and determine whether the sender is a trusted and authorized party. In this way, digital signatures support the identification and authorization processes.

Second, a digital signature assures a recipient that the owner of the enclosed certificate sent the message. A digital signature is produced using the sender’s private key. If a recipient can successfully “decode” the signature with the public key from the sender’s certificate, the recipient is positively assured that the message is from the person or organization identified on that certificate. This characteristic provides both authentication (the sending party is who it claims to be) and nonrepudiation (the sending party cannot deny issuing the message).

Finally, a digital signature assures the integrity of the message with a message digest—a hash code that is mathematically derived from the message itself. When a recipient opens a signed message, it recalculates the hash code and compares its result to the original hash code in the signature. If the values don’t match, the recipient knows that the message was deliberately or inadvertently altered after it was signed.

**Explicit and Implicit Signatures**

There are two types of digital signatures: explicit signatures and implicit signatures.

An explicit signature is appended as a separate body part to the end of a MIME message. This format is sometimes referred to as the clear-signing or detached-signature format. When a MIME entity contains an explicitly signed message, its Content-Type header field is set to “multipart/signed.” This field also specifies the protocol and message-integrity algorithm (micalg) used to produce the signature.

SAP BC Server uses the “pkcs7-signature” protocol and the “SHA-1” integrity algorithm.

**Note:** SAP BC Server automatically sets the Content-Type header field when you sign a message using the S/MIME services. Your service does not need to do this.
What Is S/MIME?

The following is an example of an explicitly signed MIME message. Notice that the message has two body parts: the first part contains the payload; the second part contains the signature.

**An explicitly signed message**

```plaintext
Date: Mon, 08 Jan 2001 10:35:23 -0500
From: "Exprint Estimating" <EXPEst@exprint.com>
To: "Purch01@GSX.com" Purch01@GSX.com
MIME-Version: 1.0
Message-ID: <36898002,DBDJ8097@exprint.com>
Content-Type: multipart/signed;
    micalg-SHA-1; protocol="application/pkcs7-signature";
    boundary="X----B164240404----X"
--X----B164240404----X

Content-Type: text/plain
Content-Transfer-Encoding: 7bit
Dear Buyer,
EXP Printing has received your request for an estimate. You will receive a formal quote within 24 hours. Your request number is RFQ-OA000011318. Questions? Contact customer service from 8:00am and 6:00pm ET at 800-334-2517.

--X----B164240404----X

Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s
MIAGCSqSIb3DQEHACAQExCAJBgUrDgMCGUAMIGCSqGSIb3QHAAoJL7I1CBBK
B7qwr+MiID56ADAqECAhAhCcC criticism P2K95hng8QMV04rMA0fGCSEemb3SIGEBBmr
.
.
Sy/fCuYadgb1JAAnuGSEy=nQVBolVRTUUrQGH5KKmTmqJbiAMQANJyTVR
8kh=

--X----B164240404----X--
```

A message can also be implicitly signed. When you use this technique, the message is encoded within the signature block, thus preventing the message from being extracted or read unless the signature is processed by a PKCS-enabled recipient. For this reason, explicit signatures are preferred, because they make the message available to non-PKCS recipients, too.

When a MIME entity contains an implicitly signed message, it’s Content-Type header field is set to “application/pkcs7-mime.”
The following is an example of a text message that has been implicitly signed. As you can see, the text of the message is not visible.

**An implicitly signed message**

```
Date: Mon, 08 Jan 2001 10:35:23 -0500
From: "Exprint Estimating" <EXPEst@exprint.com>
To: "Purch01@GSX.com" Purch01@GSX.com
MIME-Version: 1.0
Message-ID: <36898002,DBDJ8097@exprint.com>
Content-Type: application/pkcs7-mime; smime-type=signed-data;
    name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m

MIAGCSqSIb3DQEHAQEEcCAJBgUrDgMCGUAMIGCSqGSIb3QEHAoIJKJ1CCbk
B7Qw8gR+MIID6eAgECAAhMaCcpPAF01ZkZ4MA0GCSUEBwYJKwYB
Sy-F0CuYagb1JAANuGSYIE+nQVBoiVRUFUfQH5KkKmTRPjYT
9u9yKQeMggfMID6eAdeAECAhMcCcpPAF01ZkZ4MA0GCSUEB
MHITkE8vYadgb1b8c1VRUFUfQH5KkKmTRPjYT
.
.
AIhvcNAQcCoIIJozCMIIJsgYJKoCCZ8CAAFE2CggCzAjBqExMCGUAMAsGCqSGIb
P29k5hggB7Qw8gR+MIID6eAgECAAhMaCcpPAF01ZkZ4MA0GCSUEB
dgb1JAANuGSYIE+nQVBoiVRUFUfQH5KkKmTRPjYT
CAAhMcCcpPAF01ZkZ4KQeMggfMID6eAdeAECAhMaCcpPA
6hk=
```

**Encryption**

Encryption is a way to ensure privacy by assuring that a message can be read only by the intended recipient.

Encryption is performed using a pair of keys. The sending party encrypts the message using the recipient's public key. The recipient decrypts the message with its private key. Since the owner of the public key is the only one in possession of the private key, only the owner can successfully decrypt the message.

SAP BC Server supports RC2, TripleDES and DES encryption algorithms. RC2 lets you specify a key length of 40, 64, or 128. TripleDES uses a key length of 192. DES uses a key length of 64 (in US versions of the product) or 40 (in non-US versions of the product).
What Is S/MIME?

The following is an example of an encrypted message. Note that its Content-Type header field is set to “application/pkcs7-mime” (required for encrypted messages), and that the payload contains the encrypted message.

An encrypted message

```
Date: Mon, 08 Jan 2001 10:35:23 -0500
From: "Exprint Estimating" <EXPEst@exprint.com>
To: "Purch01@GSX.com" Purch01@GSX.com
MIME-Version: 1.0
Message-ID: <36898002,DBDJ8097@exprint.com>
Content-Type: application/pkcs7-mime;
   smime-type=enveloped-data; name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m

CHN8aAsGCqGIGicNAQCoIIJozCCZ8CAQEZCAzAJBqUQdGQyMDU0MC04NzIwNDQ3MDg4OC0wNDE0MjAzNQ==
```

Note: Although encryption protects a message from being read by an unintended party, it does not assure message integrity, and does not provide authentication or nonrepudiation. These qualities are guaranteed by digital signatures.
certificates to you, for example). Then, they store the certificates in their file system, a database, or a special repository for security information.

It does not make any difference where you maintain the certificates of the parties with whom you want to exchange encrypted messages, as long as the certificates are in X.509 format and can be retrieved by SAP BC Server at run time.

The MIME and S/MIME Services

The MIME and S/MIME services allow you to build secure MIME objects that you can send over the Internet. They also allow you to extract information from MIME messages that are placed in the pipeline, decrypting that information when necessary.

Services Used to Construct MIME and S/MIME Messages

The following table lists services that you use to create MIME messages and optionally secure them using a digital signature and/or encryption. For information about how you use these services to create various kinds of MIME messages, see “Building MIME and S/MIME Messages” starting on page 473.

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createMimeData</td>
<td>Creates an empty MIME object, which you use to compose a MIME message.</td>
</tr>
<tr>
<td>addMimeHeader</td>
<td>Adds one or more header fields to a MIME object.</td>
</tr>
<tr>
<td>addBodyPart</td>
<td>Adds a body part (headers and content) to a specified MIME object.</td>
</tr>
<tr>
<td>getEnvelopeStream</td>
<td>Generates a MIME message from a MIME object.</td>
</tr>
<tr>
<td>createSignedData</td>
<td>Digitally signs a MIME message.</td>
</tr>
<tr>
<td>createEncryptedData</td>
<td>Encrypts a MIME message.</td>
</tr>
<tr>
<td>createSignedAndEncryptedData</td>
<td>Digitally signs a MIME message and then encrypts it.</td>
</tr>
</tbody>
</table>

Services Used to Extract Data from MIME and S/MIME Messages

The following table lists services that you use to extract data from a MIME message. For information about how you use these services to decrypt, authenticate, and extract information from a MIME message, see “Extracting Data from MIME and S/MIME Messages” starting on page 492.
MIME Messages, MIME Entities and MIME Objects

In this book, the term MIME message refers to a complete, top-level MIME message that is made up of a set of message header fields (including the mandatory MIME Version header) and a body.

The term MIME entity refers to any block of data composed of header fields and a body. It can mean either a complete MIME message or a single body part within a multipart message.

Most MIME services provided by SAP BC Server do not operate directly on a MIME message or a MIME entity. Instead, they operate on a MIME object. A MIME object is a parsed representation of a MIME message that allows SAP BC services to add and/or retrieve the message’s constituent elements (header fields and content). By convention, the variable that holds a MIME object is called mimeData.

Building MIME and S/MIME Messages

To construct a MIME message with SAP BC Developer, you first create an “empty” MIME object and then populate the object with the appropriate header fields and content. After putting the required data into the MIME object, you generate a MIME message from the MIME object.
CHAPTER 19 Working with MIME and S/MIME Messages

The following diagram illustrates this process.

### How to construct a MIME message

1. Create an empty MIME object...

   ![Diagram showing how to create a MIME message]

2. Populate the MIME object with header fields and content, and then...

3. Generate a MIME message from the fully populated MIME object.

### Creating a MIME Message

To create a MIME message, you use services from the pub.mime folder. It contains services you use to create an empty MIME object, populate the MIME object with header fields and content, and generate the finished MIME message.

### How to Create a MIME Message

The following procedure describes the general steps you take to create a MIME message.

1. **Create an empty MIME object using** `pub.mime:createMimeData`. You do not need to pass any input parameters to this service.

   This service returns an empty MIME object named `mimeData`.

2. **Add application-specific message headers with** `pub.mime:addMimeHeader`. If your message requires application-specific (e.g., "X-type" fields) or transport-specific message headers (e.g., "To" and "From" header fields), use `addMimeHeader` to specify them. This service takes as input a Record called `mimeHeader`, whose fields and values specify message header field names and values.

---

**Diagram Description:**

- **Step 1:** Create an empty MIME object using `pub.mime:createMimeData` service.
- **Step 2:** Populate the MIME object with header fields and content.
- **Step 3:** Generate a MIME message from the fully populated MIME object.
For example, a *mimeHeader* set as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>To</td>
<td>&quot;xprint mEstimating&quot;<a href="mailto:EXPEst@exprint.com">EXPEst@exprint.com</a></td>
</tr>
<tr>
<td>From</td>
<td>&quot;<a href="mailto:Purch01@GSX.com">Purch01@GSX.com</a>&quot;<a href="mailto:Purch01@GSX.com">Purch01@GSX.com</a></td>
</tr>
<tr>
<td>X-Doctype</td>
<td>RFQ</td>
</tr>
<tr>
<td>X-Severity</td>
<td>5</td>
</tr>
</tbody>
</table>

Would produce the following message header fields:

To: "xprint Estimating"<EXPEst@exprint.com>
From: "Purch01@GSX.com"<Purch01@GSX.com>
X-Doctype: RFQ
X-Severity: 5

Note that you do *not* need to explicitly set the following message headers:

- Message-ID
- MIME-Version
- Content-Type
- Content-Transfer-Encoding

These headers are automatically generated by the service that produces the finished MIME message or are derived from parameters that you set elsewhere in your flow. If you explicitly set these fields in *mimeHeader*, they will be overwritten when the MIME message is generated.

You can set message headers before or after adding content (performing step 3, below). Either order is permitted, as long as you set them before you generate the finished MIME message.

Be aware that setting message headers before adding content will cause the headers to be included in each body part of a multipart message (i.e., in every part header in the message). To avoid this behavior, either set the message headers after you add all the body parts or drop the *mimeHeader* Record from the pipeline before you add body parts.

**Tip!** Instead of using *addMimeHeader* to add message headers, you may alternatively pass a *mimeHeader* Record to *createMimeData* when you create the MIME object.

Besides a *mimeHeader* Record, you must pass to *addMimeHeader* the *mimeData* object produced in step 1.

The *addMimeHeader* service does not return an output value. It simply updates the *mimeData* object that you pass to it.

3. **Add one or more body parts with the pub.mime:addBodyPart service.** This service adds a single body part (both header fields and content) to the MIME object. To add multiple
CHAPTER 19  Working with MIME and S/MIME Messages

body parts, execute addBodyPart once for each part that you want to add. In the finished message, body parts appear in the order in which you add them—the first body part you add will be the first body part in the message.

Besides the mimeData object that you produced in step 1, addBodyPart takes three other parameters: content, contenttype, and encoding.

— content is an InputStream containing the message content (the payload). Before invoking addBodyPart, your solution must acquire or generate this content and place it in the pipeline as an InputStream.

The way in which you acquire the content of your message depends on your particular solution. For example, you might acquire it from a file or from a back-end system. Or you might manufacture it with a custom-built service. Regardless of how you acquire your content, keep the following points in mind:

— Your content must exist as an InputStream in the pipeline. If it exists in some other form—e.g., a String or a byte[]—you must convert it to an InputStream before adding it to the MIME object.

— The InputStream should contain only the body of the message (the payload). Do not put header fields in the InputStream. To specify header fields, use the contenttype, encoding, description, and mimeHeader input parameters.

Note: If your InputStream already contains header fields, you can set the isEnvStream parameter to “true” to tell addBodyPart to pull the header fields out of the InputStream before adding it to the MIME object. For additional information about using the isEnvStream parameter, see the addBodyPart description in the SAP BC Built-In Services Guide.

— Do not encode the content before adding it to the MIME object—simply add it in its original form. If you want to encode the content for transport, set the encoding parameter (see below).

— contenttype is a String specifying the value of the entity’s Content-Type header field. Besides type and subtype, be sure to include any parameters that the header field requires as shown in the following example:

    text/plain;charset=UTF8

For a description of standard content types, see RFC 2046 - MIME Media Types at http://www.imc.org/rfc2046.

— encoding is a String specifying the value of the entity’s Content-Transfer-Encoding header field. This field also specifies the scheme in which you want the entity’s content encoded. If you set encoding to “base64,” for example, the getEnvelopeStream service will base64-encode the data in content when it generates the finished MIME message.
encoding must be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7bit</td>
<td>Specifies that content is 7-bit, line-oriented text that needs no encoding. Use this value when content contains lines of 7-bit, US-ASCII text (no octets with decimal values greater than 127; no NULs).</td>
</tr>
<tr>
<td>8bit</td>
<td>Specifies that content is 8-bit, line-oriented text that needs no encoding. Use this value when content contains lines of 8-bit text (octets with decimal values greater than 127; no NULs).</td>
</tr>
<tr>
<td>binary</td>
<td>Specifies that content contains binary information that needs no encoding. Use this value when content contains an arbitrary sequence of octets (binary data).</td>
</tr>
<tr>
<td>quoted-printable</td>
<td>Specifies that content contains 7 or 8-bit, line-oriented text that you want to encode using the quoted printable encoding scheme.</td>
</tr>
<tr>
<td>base64</td>
<td>Specifies that content contains an arbitrary sequence of octets (binary data) that you want to encode using the base64 encoding scheme.</td>
</tr>
<tr>
<td>uuencode</td>
<td>Specifies that content contains an arbitrary sequence of octets that you want to encode using the uuencode encoding scheme.</td>
</tr>
</tbody>
</table>

Note: This encoding value is not recommended for messages that will be transported via SMTP over the Internet, because intervening servers that cannot accommodate 8-bit text may alter your content. To safely transport 8-bit text, use quoted-printable encoding instead.

Note: This encoding value is not recommended for messages that will be transported via SMTP over the Internet, because intervening servers that cannot accommodate binary data may alter your content. To safely transport binary data, use base64 encoding instead.
CHAPTER 19 Working with MIME and S/MIME Messages

Note: Besides the content, contenttype, and encoding, parameters described above, the addBodyPart service has a few other optional parameters you can use. For information about these parameters, see the addBodyPart description in the SAP BC Built-In Services Guide.

The addBodyPart service does not return an output value. It simply updates the mimeData object that you pass to it.

4 Generate the finished MIME message with the pub.mime:getEnvelopeStream service. After you finish populating the MIME object, invoke getEnvelopeStream to generate a MIME message. This service takes the populated mimeData object and produces an InputStream called envStream, containing the finished MIME message.

When getEnvelopeStream generates a MIME message, it does the following:

---

- Generates the Message-ID, MIME-Version, Content-Type, and Content-Transfer-Encoding message headers and inserts them at the top of the message.
- Sets the Content-Type header to “multipart,” generates a boundary string, and inserts it between body parts if mimeData contains multiple body parts.

Note: If mimeData contains a single body part, getEnvelopeStream will, by default, create an ordinary, single-part message. Some solutions, however, want a “multipart” message even if the message contains only a single body part. If your solution requires this structure, you can use the createMultipart parameter to tell getEnvelopeStream to generate a multipart message regardless of the number of body parts it finds in mimeData.

---

- Encodes the content for each body part according to its encoding value.

Example—Creating a Single-Part MIME Message

The following flow service creates a single-part MIME message that contains a simple text message. This example is located in sample.mime:build_SimpleMIME in the WmSamples package. You can open this example with Developer to see how the pipeline is mapped between steps.

Flow Service that creates a simple MIME message

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>CreateMimeData (create an empty MIME object)</td>
</tr>
<tr>
<td>1.2</td>
<td>addMimeHeader (add any application-specific headers, e.g., &quot;X&quot;-type headers)</td>
</tr>
<tr>
<td>2.1</td>
<td>stringToListStream (get content as InputStream)</td>
</tr>
<tr>
<td>2.2</td>
<td>addBodyPart (add content to MIME object)</td>
</tr>
<tr>
<td>3.1</td>
<td>getEnvelopeStream (generate MIME message)</td>
</tr>
<tr>
<td>3.2</td>
<td>StreamToText (convert finished MIME message to String for viewing)</td>
</tr>
</tbody>
</table>
Step | Description
--- | ---
1 | This step creates an empty MIME object. It does not take any inputs. It puts an empty MIME object named mimeData in the pipeline.
2 | This step adds two application-specific message headers in the MIME object. If you view the pipeline, you will see that the mimeHeader input variable is set as follows:
   | **Name** | **Value**
   | X-DocType | alert
   | X-Severity | 9
3 | This step generates the content of the message. This example uses a custom Java service to convert a String containing the following text to an InputStream:
   
   We were not able to process your request because the account number you gave us has expired. Please correct the account number and resubmit your request.

   In practice, you are more likely to acquire your content from a file, the network, or a back-end system.
4 | This step adds the content produced by step 3 to the mimeData object. If you view the pipeline, you will note that the stream output variable from step 3 is mapped to this step's content input variable. Because content contains a simple text message, the contenttype and encoding parameters are set as follows:
   | **Parameter** | **Value**
   | contenttype | text/plain; charset=UTF8
   | encoding | quoted-printable
   | isEnvStream | is set to “no” because the payload is not a MIME entity.
This step generates the finished MIME message. It takes the `mimeData` object that was populated in steps 2 and 4 and produces an `InputStream` called `envStream` that contains the MIME message. At this point, you could pass `envStream` to any process that expects a MIME message as input.

Because you cannot view an `InputStream`, this example includes a step that converts `envStream` to a String so you can examine the finished message with Developer. This technique is useful for testing and debugging.

If you examine the contents of `string` on the **Results** tab, you will see a MIME message similar to the following:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>envStream</code></td>
<td><code>java.io.ByteArrayInputStream</code></td>
</tr>
<tr>
<td><code>MIME</code></td>
<td><code>Message-ID: &lt;1459288537.981395481288.JavaMail.gmisma@exchange.com&gt;</code></td>
</tr>
</tbody>
</table>

```
We were not able to process your request because the account number you gave us has expired. Please correct the account number and resubmit your request.
```

Example—Creating a Multipart MIME Message

The following flow service creates a multipart MIME message that contains three parts: a simple text message, an XML document, and an image file. Note that the steps you use to create a multipart message are essentially the same as the ones you use to create a single-part MIME message—the only difference is that you execute `addBodyPart` multiple times.

This example is located in `sample.mime:build_MultipartMIME` in the `WmSamples` package. You can open this example with Developer to see how the pipeline is mapped between steps.
Flow Service that creates a multipart MIME message

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This step creates an empty MIME object. It does not take any inputs. It puts an empty MIME object called <em>mimeData</em> in the pipeline.</td>
</tr>
</tbody>
</table>
| 2    | This step generates the content of the message and adds it to the *mimeData* object. If you view the pipeline for the addBodyPart service in this step, you will see that the *stream* output variable generated by the stringToStream service is mapped to the *content* input variable. Because *content* contains a simple text message, the *contenttype* and *encoding* parameters are set as follows:  
  - **Parameter**: *contenttype*  
  - **Value**: text/plain;charset=UTF8  
  - **Parameter**: *encoding*  
  - **Value**: quoted-printable |
| 3    | This step creates an XML document from a Record in the pipeline, converts that document to an InputStream, and then adds the InputStream to the *mimeData* object. Because *content* contains an XML document, the *contenttype* and *encoding* parameters are set as follows:  
  - **Parameter**: *contenttype*  
  - **Value**: text/xml  
  - **Parameter**: *encoding*  
  - **Value**: quoted-printable |
| 4    | This step gets an image file from disk and adds it to the *mimeData* object. Because the file is retrieved as an InputStream, it can be mapped directly to the *mimeData* object. In this case, *content* is an image file (binary data), so the *contenttype* and *encoding* parameters are set as follows:  
  - **Parameter**: *contenttype*  
  - **Value**: image/gif;name="b2b.gif"  
  - **Parameter**: *encoding*  
  - **Value**: base64 |
Points to keep in mind when building multipart MIME messages:

- By default, the Content-Type header field is set to “multipart/mixed.” If you want to use a different subtype, set the `subtype` parameter when you invoke `createMimeData`.

- Body parts appear in the message in the order in which you add them to the MIME object—the first part you add appears first in the message.
If you set message headers (e.g., using addMimeHeader) before you add body parts, those header fields will also be inserted into each body part. If you do not want this to occur, drop the mimeHeader variable from the pipeline before you perform an addBodyPart step or execute the addMimeHeader step after adding the message's body parts.

**Signing a MIME Message**

To digitally sign a MIME message you must have access to the following credentials:

- The signer’s private key
- The signer’s certificate
- The signer’s certificate chain. If you know that the recipient trusts an intermediate CA in your chain, you can supply a partial chain that extends back to that CA. However, if you are not sure which CA the recipient trusts, supply a complete chain.

**Note:** You are not required to have the signer’s certificate chain to sign a message; however, if you omit the chain, the recipient must produce the certificate chain when it receives the message. If you do not supply the signer’s certificate chain, and the recipient does not have a local copy of it, the signature verification process will fail. By including the certificate chain with a signature, you ensure that the recipient will be able to process the signature.

Many sites maintain these credentials in files on their SAP BC Server. (If your server uses SSL, the server’s certificates will reside somewhere on your file system in a set of .DER files). Some sites store them in a database or a special repository for security information. It does not make any difference where the credentials are stored, as long as they are in X.509 format and can be retrieved by SAP BC Server at run time.

If you cannot locate these credentials or do not have direct access to them, consult your SAP BC Server Administrator.

**How to Create a Signed S/MIME Message**

The following procedure describes the general steps you take to create a signed S/MIME message.

**Important!** If you want to create a signed and encrypted MIME message, use the special service that SAP provides for this purpose. For more information, see “Signing and Encrypting a MIME Message” on page 489.
1 Create an InputStream that contains the MIME message that you want to sign. Use the procedure outlined in “Creating a MIME Message” on page 474 to create the MIME message.

2 Fetch the signer’s private key, certificate, and certificate chain. These credentials must reside in the pipeline in the following form:

<table>
<thead>
<tr>
<th>Credential</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private key</td>
<td>A byte array containing the signer’s private key.</td>
</tr>
<tr>
<td>Certificate</td>
<td>A byte array containing the signer’s certificate.</td>
</tr>
<tr>
<td>Certificate chain</td>
<td>A list (a one-dimensional array) of byte arrays containing the signer’s complete certificate chain, where element 0 in the list contains the signer’s certificate and element 1 contains the CA’s certificate. If the certificate in element 1 is an intermediate CA that is not trusted by the recipient, successive elements must contain the certificates of the other intermediaries in the chain (in order), extending back to a trusted intermediary or the root-level CA.</td>
</tr>
</tbody>
</table>

For example, a chain made up of the owner, two intermediaries, and the root-level CA would look as follows:

<table>
<thead>
<tr>
<th>Element #</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Signer’s certificate</td>
</tr>
<tr>
<td>1</td>
<td>Intermediary CA Certificate</td>
</tr>
<tr>
<td>2</td>
<td>Intermediary CA Certificate</td>
</tr>
<tr>
<td>3</td>
<td>Root CA Certificate</td>
</tr>
</tbody>
</table>

The way in which you fetch these credentials depends on where they are stored at your site. If they are stored in the file system, you can retrieve them using the pub.file:getFile service. If they are stored in a special repository or a database system, you may need to develop a custom service to retrieve them.

3 Pass the credentials and the MIME message to the pub.smime:createSignedData service. This service takes an InputStream containing a MIME message and signs it using the private key and the certificates that you provide. It produces an InputStream containing the signed message.

Example—Signing a MIME Message

The following flow service signs a single-part MIME message. This example resides in sample.smime:build_SignedSMIME in the WmSamples package. You may want to open this example with Developer to see how the pipeline is mapped between steps.
To run this example, you must have a private key, its associated certificate, and the certificate of the CA who signed it. When you run this service from Developer, it will prompt you for the following:

**Input Parameter**  
**Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>signersPrivateKeyFile</td>
<td>The name of the file containing the signer’s private key, for example, d:\certs\myprivkey.der.</td>
</tr>
<tr>
<td>signersCertificateFile</td>
<td>The name of the file containing the certificate that belongs to signersPrivateKeyFile, for example, d:\certs\mycert.der.</td>
</tr>
<tr>
<td>signersCACertificateFile</td>
<td>The name of the file containing the certificate of the CA that issued signersCertificateFile, for example, d:\certs\myCAcert.der.</td>
</tr>
</tbody>
</table>

**Flow Service that signs a MIME message**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build_SignS MIME</td>
<td></td>
</tr>
<tr>
<td>acquireSeq ( )</td>
<td>STEP 1 - Generate the MIME message</td>
</tr>
<tr>
<td>goFile ( )</td>
<td>STEP 2 - Get signer’s private key</td>
</tr>
<tr>
<td>getFile ( )</td>
<td>STEP 3 - Get signer’s certificate</td>
</tr>
<tr>
<td>loadCertificateObjectList ( )</td>
<td>STEP 4 - Build signer’s certificate chain</td>
</tr>
<tr>
<td>createSignedData ( )</td>
<td>STEP 5 - Sign the MIME message</td>
</tr>
<tr>
<td>streamToString ( )</td>
<td>STEP 6 - FOR DEBUG ONLY. Convert finished SMIME message to String for viewing</td>
</tr>
</tbody>
</table>

**Step**  
**Description**

1. This step creates a MIME message containing a simple text message. It produces an InputStream called *envStream* that contains the MIME message that will be signed.

2. This step retrieves the signer’s private key from the file specified in signersPrivateKeyFile. Note that the loadAs flag is set to “bytes” to load the file into the pipeline as a byte[ ].

3. This step retrieves the signer’s certificate from the file specified in signersCertificateFile. Note that the loadAs flag is set to “bytes” to load the file into the pipeline as a byte[ ].

4. This step builds the certificate chain from the files specified in signersCertificateFile and signersCACertificateFile. This example uses a custom Java service to perform this step. You will need to develop a similar mechanism to assemble a chain based on where your certificates are located and the number of certificates in the chain.

5. This step generates the signed MIME message. It takes the InputStream from step 1 and the credentials loaded in steps 2–4 and produces an InputStream called SMimeEnvStream that contains the signed message.
Because you cannot view the contents of an InputStream, this example includes a step that converts S/MIME messages to a String so you can examine the finished message with Developer. This technique is useful for testing and debugging.

If you examine the contents of string on the Results tab, you will see a signed S/MIME message similar to the following. Note that this example creates an explicitly signed message—the message is in one body part and the digital signature is in another.

```
Message-ID: <1168925611.910.1000013519.9910222025.1.80001.1.1.1@research.microsoft.com>

Message-ID: <1168925611.910.1000013519.9910222025.1.80001.1.1.1@research.microsoft.com>

Content-Type: multipart/mixed; protocol="application/octet-stream"; boundary="="

--MIME_boundary
Content-Type: text/plain; charset=US-ASCII
Content-Transfer-Encoding: quoted-printable

We were not able to process your request because the account number you gav- e us has expired. Please correct the account number and resubmit your reques- t.

--MIME_boundary
Content-Type: application/octet-stream; name=signature.p7s
Content-Transfer-Encoding: base-64
Content-Disposition: attachment; filename=signature.p7s
```
Encrypting a MIME Message

To encrypt a MIME message you must have the recipient's certificate. It contains the public key required to encrypt the message. To obtain this certificate, ask the parties with whom you want to exchange encrypted messages to send you their X.509 digital certificate in DER file format. For example, you might ask a party to e-mail the .DER file to you or transmit it in a special “certificates only” MIME message, which you can read using the pub.smime:processCertsOnlyData service.

After receiving the certificate, store it in a location (file system, database, or other repository) where it can be retrieved by SAP BC Server at run time. Once you have access to the recipient's certificate, you can generate encrypted MIME messages that only the owner of the certificate can read.

How to Create an Encrypted S/MIME Message

The following procedure describes the general steps you take to create an encrypted S/MIME message.

Important! If you want to create a signed and encrypted MIME message, use the special service that SAP provides for this purpose. For instructions, see “Signing and Encrypting a MIME Message” on page 489.

1. Create an InputStream containing the MIME message that you want to encrypt. You can use the procedure outlined in “Creating a MIME Message” on page 474 to create the MIME message.
2. Fetch the recipient's certificate as a byte[]. If the message will be sent to multiple recipients, fetch the certificate of each recipient. Load the certificates into a list (a one-dimensional array) of byte[] such that each element in the list holds the certificate of single recipient.
3. Pass the certificate and the MIME message to the pub.smime:createEncryptedData service. This service encrypts the InputStream containing the MIME message and produces a new InputStream containing the encrypted message.

Example—Encrypting a MIME Message

The following flow service encrypts a MIME message. This example is located in sample.smime:build_EncryptedSMime in the WmSamples package. You can open this example with Developer to see how the pipeline is mapped between steps.

To run this example, you must have at least one certificate file. When you run this service from Developer, it will prompt you for the following:
Flow Service that encrypts a MIME message

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This step creates a MIME message containing a simple text message. It produces an InputStream called <code>envStream</code> that contains the MIME message that will be encrypted.</td>
</tr>
<tr>
<td>2</td>
<td>This step loads the recipient’s certificates from the files specified in <code>recipient1CertificateFile</code> and <code>recipient2CertificateFile</code>. This example uses a custom Java service to perform this step. You will need to develop a similar mechanism to load the certificates of the parties to whom you want to send an encrypted message.</td>
</tr>
<tr>
<td>3</td>
<td>This step generates the encrypted MIME message using the InputStream from step 1 and the certificates from step 2. It produces a new InputStream called <code>SMimeEnvStream</code> that contains the encrypted message.</td>
</tr>
<tr>
<td>4</td>
<td>Because you cannot view the contents of an InputStream, this example includes a step that converts <code>SMimeEnvStream</code> to a String so you can examine the finished message with Developer. This technique is useful for testing and debugging. If you examine the contents of <code>string</code> on the Results tab, you will see an encrypted S/MIME message similar to the following:</td>
</tr>
</tbody>
</table>
Signing and Encrypting a MIME Message

If you want to sign and encrypt a message, you must use the special service, pub.smime:createSignedAndEncryptedData, that SAP provides for this purpose.

To use this service, you need the credentials for signing a message (the signer’s private key, certificate, and certificate chain) and the credentials needed to encrypt a message (the recipient’s certificate). For information about obtaining these credentials, see “Signing a MIME Message” on page 483 and “Encrypting a MIME Message” on page 487.

Example—Signing and Encrypting a MIME Message

The following flow service signs and encrypts a MIME message. This example resides in sample.smime:build_SignedAndEncryptedSMime in the WmSamples package. You can open this example with Developer to see how the pipeline is mapped between steps.

To run this example, you must have the private key, the associated certificate, and the certificate of the CA that issued it.

When you run this service from Developer, it will prompt you for the following:
CHAPTER 19  Working with MIME and S/MIME Messages

Input Parameter | Description
--- | ---
signersPrivateKeyFile | The name of the file containing the signer’s private key. For example, d:\certs\myprivkey.der.
signersCertificateFile | The name of the file containing the certificate that belongs to signersPrivateKeyFile, for example, d:\certs\mycert.der.
signersCACertificateFile | The name of the file containing the certificate of the CA that issued signersCertificateFile, for example, d:\certs\myCAcert.der.
recipient1CertificateFile | The name of the file containing the certificate of the first intended recipient, for example, d:\netCerts\partner1cert.der.
recipient2CertificateFile | The name of the file containing the certificate of the second intended recipient. If you want to encrypt the message for only one recipient, leave this input parameter empty.

Flow Service that signs and encrypts a MIME message

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This step creates a MIME message that contains a simple text message. It produces an InputStream called envStream, containing the MIME message that will be signed and encrypted.</td>
</tr>
<tr>
<td>2</td>
<td>This step loads the credentials used to sign the message. It performs the following steps:</td>
</tr>
<tr>
<td></td>
<td>1) It retrieves the signer’s private key from the file specified in signersPrivateKeyFile. Note that the loadAs flag is set to “bytes” to load the file into the pipeline as a byte[].</td>
</tr>
<tr>
<td></td>
<td>2) It retrieves the signer’s certificate from the file specified in signersCertificateFile. Note that the loadAs flag is set to “bytes” to load the file into the pipeline as a byte[].</td>
</tr>
</tbody>
</table>
It builds a certificate chain from the files specified in `signersCertificateFile` and `signersCACertificateFile`. This example uses a custom Java service to perform this step. You will need to develop a similar mechanism to assemble a chain based on where your certificates are located and the number of certificates in the chain.

This step loads the recipient's certificates from the files specified in `recipient1CertificateFile` and `recipient2CertificateFile`. This example uses a custom Java service to perform this step. You will need to develop a similar mechanism to load the certificates of the parties to whom you want to send an encrypted message.

This step generates the signed and encrypted MIME message. It takes the InputStream from step 1 and the credentials from steps 2–5 and produces an InputStream called `SMimeEnvStream` containing the signed and encrypted message.

Because you cannot view the contents of an InputStream, this example includes a step that converts `SMimeEnvStream` to a String so you can examine the finished message with Developer. This technique is useful for testing and debugging.

If you examine the contents of `string` on the Results tab, you will see an encrypted S/MIME message similar to the following:

```
This is an S/MIME message. It contains two encrypted attachments:

1. A signed and encrypted PDF file.

The message is signed with the sender's private key and encrypted with the recipient's public key.
```

### Table: build_SignedAndEncryptedS/MIME

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMimeEnvStream</td>
<td>java.io.InputStream</td>
</tr>
<tr>
<td>Message-ID</td>
<td>8279813204:9824690602512:JavaMail</td>
</tr>
<tr>
<td>Mime-Version</td>
<td>1.6</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/pem-x509-signedité</td>
</tr>
<tr>
<td>Content-Disposition</td>
<td>attachment=x509-signature</td>
</tr>
<tr>
<td>SMimeEnvStream</td>
<td>java.io.InputStream</td>
</tr>
<tr>
<td>Message-ID</td>
<td>8279813204:9824690602512:JavaMail</td>
</tr>
<tr>
<td>Mime-Version</td>
<td>1.6</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/pem-x509-signedité</td>
</tr>
<tr>
<td>Content-Disposition</td>
<td>attachment=x509-signature</td>
</tr>
</tbody>
</table>

The message contains two attachments:

1. A signed and encrypted PDF file.
Extracting Data from MIME and S/MIME Messages

Besides creating MIME messages, you can also use SAP BC services to extract information—header fields and/or content—from MIME messages that are placed in the pipeline. However, to gain access to the data within a MIME message, you must first convert that message to a MIME object. After you convert the message to a MIME object, you use services such as getMimeHeader, getBodyPartContent, and getBodyPartHeader, to retrieve the information within it.

The following diagram illustrates this process:

Extracting Data from a MIME message

1. Acquire the MIME message and put it in the pipeline.
2. Generate a MIME object from the MIME message.
3. Use the pub.mime services to extract the data.

Extracting the Payload from a MIME Message

To extract information from a MIME message, you use services from the pub.mime folder. It contains the services you use create a MIME object from a MIME message and extract data from it.

How to Extract the Payload from a MIME Message

The following procedure describes the general steps you take to extract data from a MIME message.

1. Place the MIME message in the pipeline as an InputStream. For the SAP BC services to work with a MIME message, the message must be passed into the pipeline as an InputStream. If your solution acquires the MIME message in another form—such as a String or a byte[]—you must convert it to an InputStream before running the MIME services against it.
2 Convert the MIME message to a MIME object using the pub.mime:createMimeData service.
   Pass the InputStream containing the MIME message to createMimeData. This service returns a MIME object called mimeData that contains the message's constituent elements (header fields and content). It also returns a set of values indicating whether the enclosed message is encrypted or digitally signed. (For information about extracting information from an encrypted and/or signed MIME message, see “Extracting the Payload from a Signed MIME Message” on page 496 and “Extracting the Payload from an Encrypted MIME Message” on page 500.)

3 Extract the payload from the MIME object using the pub.mime:getBodyPartContent service.
   This service takes as input the MIME object that you created in step 2. If the message contains multiple parts, you can use the index or contentID parameter to specify which part you want to retrieve, where:

   — index is a String that specifies the index number (i.e., position number) of the part whose content you want to retrieve. (Index 0 is the first part, Index 1 is the second part, and so forth.)

   — contentID is a String that specifies the value of the content-ID header whose content you want to retrieve. For example, if you wanted to retrieve the content of the part with the “content-ID: AJ9994388-0500,” you would set contentID to “AJ9994388-0500.”

If you don’t specify index or contentID, getBodyPartContent returns the content from the first body part in the message.

The content of the requested body part is returned as an InputStream named content.
Example—Extracting One Part from a Multipart MIME Message

The following flow service shows how you would extract the content from the second body part in a three-part MIME message. In this example, the message contains an XML document that is extracted, parsed, and put in the pipeline.

This example is located in sample.mime:extract_SimpleMIME in the WmSamples package. You can open this example with Developer to see how the pipeline is mapped between steps.

Flow Service that extracts content from a single part

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This step acquires a MIME message. This example calls a helper service that puts a three-part test message in the pipeline as an InputStream. In a production solution, it is more likely that a MIME message would be passed into the pipeline by a content handler or a back-end system.</td>
</tr>
<tr>
<td>2</td>
<td>This step takes the MIME message and creates a MIME object (mimeData) containing the message’s headers and content. If you view the pipeline, you will note that the InputStream produced by step 1 is mapped to this step’s input variable.</td>
</tr>
<tr>
<td>3</td>
<td>This step extracts the payload from the second body part in mimeData. In this example, the index parameter is set to 1 to select the second body part. This step returns the payload (in this case an XML document) as an InputStream named content.</td>
</tr>
<tr>
<td>4</td>
<td>This step converts the XML document in content to a String.</td>
</tr>
<tr>
<td>5</td>
<td>This step takes the String containing the XML document and parses it, producing a Document object (a node) containing the XML document.</td>
</tr>
<tr>
<td>6</td>
<td>This step produces a Record containing the XML document’s elements and values.</td>
</tr>
</tbody>
</table>
Example—Extracting All Parts from a Multipart MIME Message

The following flow service shows how you might process each part in a multipart MIME message sequentially. This example receives a multipart MIME message containing an unknown number of body parts. After discovering the number of body parts, the example uses a REPEAT block to extract the payload from each part.

This example is located in sample.mime:extract_MultipartMIME in the WmSamples package. You can open this example with Developer to see how the pipeline is mapped between steps.

Flow Service that extracts the content from multiple parts

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This step acquires a MIME message. This example uses a helper service that generates a three-part MIME message and puts it in the pipeline as an InputStream call envStream. In a production solution, it is more likely that a MIME message would be passed into the pipeline by a content handler or a back-end system.</td>
</tr>
<tr>
<td>2</td>
<td>This step takes the MIME message and creates a MIME object (mimeData) containing the message’s headers and content. If you view the pipeline, you will note that the InputStream produced by step 1 is mapped to this step’s input variable.</td>
</tr>
<tr>
<td>3</td>
<td>This step inspects the mimeData object and returns the number of body parts (in this case 3) in a String called numParts.</td>
</tr>
<tr>
<td>4</td>
<td>This step sets the following variables that are used by the REPEAT block:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>RepeatCounter</td>
<td>numParts-1</td>
<td>Sets the counter that specifies the number of times the REPEAT block needs to re-execute. Since a REPEAT block always executes once, this counter is set to one number less than the total number of body parts in the message.</td>
</tr>
<tr>
<td>PartIndex</td>
<td>0</td>
<td>Initializes the pointer that is used to step through the body parts in this message.</td>
</tr>
</tbody>
</table>
CHAPTER 19  Working with MIME and S/MIME Messages

Extracting the Payload from a Signed MIME Message

When you pass a signed S/MIME message to createMimeData, it returns an empty MIME object, because it cannot parse signed messages. To extract data from a signed message, you must process the message with pub.smime:processSignedData. This service reads an InputStream containing a signed message, verifies the signature, and returns a MIME object containing the message’s constituent elements.

Important! A signer’s certificate is authenticated against the set of trusted certificates in SAP BC Server’s CA certificate directory. If your site will receive signed messages, you must collect the certificates of CAs that you trust, store them in a directory on the file system, and then set the server’s CA Certificate Directory parameter to point to this directory. For information about setting this parameter, see the SAP BC Administration Guide.

How Do You Know Whether the Message Is Signed?

If your solution always receives signed messages, you can simply pass those messages to processSignedData when you receive them. However, if your solution receives both signed and unsigned messages, you will need to “test” the message to see whether or not it is signed and pass only signed messages to the processSignedData service.

To discover whether a MIME message is signed, pass it to the createMimeData service and check the status of the signed variable afterwards. If the value of signed is “true,” you must pass the message to processSignedData for signature verification.

Working with InputStreams

To work with signed (and encrypted) messages successfully, you need to understand something about the behavior of an InputStream. InputStreams are transient forms of data in a flow service. When a service reads an InputStream, it immediately discards the data within it. This means that once you process a MIME message with createMimeData, the message no longer exists in the original InputStream object.
This poses a problem if, after running `createMimeData` on the InputStream, you discover that it contains a signed or encrypted message. Since the original InputStream has been emptied, it cannot be passed to the signature-verification or decryption services. To solve this problem, `createMimeData` returns a copy of the original InputStream in an output variable called `stream`. This is the variable you pass to `processSignedData` if, after processing the original message with `createMimeData`, you discover that it is signed.

**What Happens when the Signature is Processed?**

When `processSignedData` processes a signed message, it does the following:

- It verifies the digital signature using the signer’s public key.
- It compares the signer’s certificate chain to the certificates in SAP BC Server’s “trusted CA” directory to determine whether the credentials are authentic and trustworthy.
- It extracts the message from the S/MIME message stream, parses it, and puts it in the pipeline as a MIME object called `mimeData`.

**Error Codes and Messages**

If an error prevents the signature from being verified—for example, if the signer’s certificate cannot be read or the signature itself is found to be invalid—`processSignedData` sets the `verify` flag to false and reports the cause of the failure in `errorCode` and `errorMessage` as follows:

| `errorCode` | `errorMessage` | Signature could not be verified because...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Invalid signer certificate file information.</td>
<td>The signer’s certificate could not be read. The variable containing the certificate chain is not an array object.</td>
</tr>
<tr>
<td>2</td>
<td>Certificate at index ‘i’ is not in recognizable format.</td>
<td>The signer’s certificate could not be read. The data at position i in the certificate chain does not appear to be a certificate.</td>
</tr>
<tr>
<td>3</td>
<td>Invalid certificate input at index ‘i’.</td>
<td>The signer’s certificate could not be read. The data at position i in the certificate chain is not a byte[].</td>
</tr>
<tr>
<td>4</td>
<td>Signature cannot be verified.</td>
<td>The signature was invalid. Either the supplied certificate does not belong to the original signer or a message integrity violation has occurred.</td>
</tr>
</tbody>
</table>

If `processSignedData` is able to verify the signature, but is not able to authenticate the certificate of the signer (i.e., the certificate could not be confirmed to be from a trusted source), the `verify` flag will be true and the `errorCode` and `errorMessage` values will be set as follows:
CHAPTER 19  Working with MIME and S/MIME Messages

If processSignedData is able to verify the signature and authenticate the signer's certificate, it does not return errorCode or errorMessage.

Note: Regardless of whether processSignedData is able to verify the signature or authenticate the certificate, it always returns a MIME object containing the parsed message.

How to Extract the Payload from a Signed S/MIME Message

The following procedure describes the general steps you take to extract data from a signed S/MIME message.

1  If you do not know whether the message is signed, pass it to the pub.mime:createMimeData service. Afterwards, test the state of the signed output parameter. If its value is “true,” proceed to step 2. Otherwise, check whether the message is encrypted and process it as described in “Extracting the Payload from an Encrypted MIME Message” on page 500. If the message is neither signed nor encrypted, process it as an ordinary MIME message as described in “Extracting the Payload from a MIME Message” on page 492.

2  Pass the message to the pub.smime:processSignedData service to verify the signature. If the signer’s certificate chain is included in the signature, you do not need to give this service anything other than the InputStream containing the MIME message. If the signer’s certificate chain is not embedded in the signature, you must supply it (this assumes that the signer has given you a certificate chain at some point).

   Keep in mind that if the message was passed to createMimeData before this step, the original InputStream will be empty. In this case, you must pass the stream output variable produced by createMimeData to the processSignedData service.

3  Test the verify flag and perform error processing as necessary. If the signature cannot be verified, verify will be false. Your service should contain logic to detect this condition and react in a prescribed way. For example, it might send the message to an administrator for a manual inspection or record the event in a log file.
Note: Depending on the nature of the messages your service receives, you may want to test the encrypted output variable after processing a signature. This will tell you whether the message had been encrypted before it was signed. If encrypted is “true,” you will need to decrypt the message in stream. For procedures, see “Extracting the Payload from an Encrypted MIME Message” on page 500.

4 Extract the payload from the MIME object using the pub.mime:getBodyPartContent service. If the enclosed message is not encrypted, processSignedData returns a MIME object that contains the message’s constituent elements (header fields and content). At this point, you can use getBodyPartContent to retrieve the content from the MIME object. For information about using getBodyPartContent, see “Extracting the Payload from a MIME Message” on page 492.

Example—Extracting Content from a Signed S/MIME Message

The following flow service extracts the payload from a signed MIME message. This example resides in sample.smime:extract_SignedSMime in the WmSamples package. You can open this example with Developer to see how the pipeline is mapped between steps.

To run this example, you must have a private key, the associated certificate, and the certificate of the CA that signed it. These credentials are needed by the helper service, sample.smime.helpers:acquireSignedMsg, which generates the signed test message used in this example. You will need to edit the first step in the helper service to specify the location of these files on your system.

This service assumes that the signature contains the signer’s certificate chain, so you do not need to supply a certificate chain at run time.

Flow Service that extracts the content from a signed MIME message

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This step acquires an InputStream containing a signed MIME message. This example uses a helper service to produce a test message. In a production solution, it is more likely that a MIME message would be passed into the pipeline by a content handler or a back-end system.</td>
</tr>
</tbody>
</table>
CHAPTER 19 Working with MIME and S/MIME Messages

2 This step takes the InputStream generated in step 1 and processes the signature. If the signature is valid, this step produces a MIME object called mimeData, containing the parsed message. If the signature is invalid, this step returns an empty mimeData object and sets the verify flag to “false.”

3 This step checks whether or not the signature was processed successfully by testing the value of the output variable verify. If verify is “true,” this step extracts the payload and converts it to a String. If verify is “false,” this step collects the error information in the pipeline and passes it to an error-logging service.

Extracting the Payload from an Encrypted MIME Message

When you pass an encrypted S/MIME message to the createMimeData service, it returns an empty MIME object, because it cannot parse encrypted messages. To extract data from an encrypted message, you must decrypt the message with pub.smime:processEncryptedData. This service reads an InputStream that contains an encrypted message, decrypts it using a private key that you supply, and returns a MIME object containing the message’s constituent elements.

How Do You Know Whether the Message Is Encrypted?

If your solution always receives encrypted messages, you can simply pass those messages to processEncryptedData when you receive them. However, if your solution receives both encrypted and clear-text messages, you will need to “test” a message to see whether or not it is encrypted, and pass only encrypted messages to the processEncryptedData service.

To discover whether a MIME message is encrypted, pass it to the createMimeData service and check the status of the encrypted variable afterwards. If the value of encrypted is “true,” you must pass the message to processEncryptedData to be decrypted.

Note: When you process an InputStream with createMimeData, that InputStream is emptied and is no longer available to other services. For this reason, createMimeData returns a copy of the original message stream in the output variable called stream. You pass this variable to processEncryptedData if the original InputStream has been emptied by createMimeData. For additional information about InputStreams, see “Working with InputStreams” on page 496.

How to Extract the Payload from an Encrypted S/MIME Message

The following procedure describes the general steps you take to extract data from an encrypted S/MIME message.

1 If you do not know whether the message is encrypted, pass it to the pub.mime:createMimeData service. Afterwards, test the state of the encrypted output parameter. If its value is “true,” proceed to step 2. Otherwise, test the signed variable to see whether the message is signed and process it as described in “Extracting the Payload from a Signed MIME Message” on page 496. If the message is neither signed nor encrypted,
Extracting Data from MIME and S/MIME Messages

process it as an ordinary MIME message as described in “Extracting the Payload from a MIME Message” on page 492.

2 **Pass the message to the pub.smime:processEncryptedData to be decrypted.** You must pass three input parameters to this service: the InputStream containing the encrypted MIME message, the recipient’s private key and the recipient’s digital certificate. The last two parameters must be in the form of byte arrays (i.e., byte[]).

Keep in mind that if the message was passed to createMimeData prior to this step, the original InputStream will be empty. In this case, you must pass the stream output variable produced by createMimeData to the processEncryptedData service.

**Note:** Depending on the nature of the messages your service receives, you may want to test the signed output variable after decrypting the message. This will tell you whether the message had been signed prior to being encrypted. If signed is “true,” you will need to verify the signature of the message in stream. For procedures, see “Extracting the Payload from a Signed MIME Message” on page 496.

3 **Extract the payload from the MIME object using the pub.mime:getBodyPartContent service.** If the decrypted message is not signed, the MIME object returned by processEncryptedData will contain the message’s constituent elements. You use getBodyPartContent to retrieve the content from this MIME object. For information about using getBodyPartContent, see “Extracting the Payload from a MIME Message” on page 492.

**Example—Extracting Content from an Encrypted S/MIME Message**

The following flow service extracts data from an encrypted multipart MIME message. This example resides in sample.smime:extract_EncryptedSMime in the WmSamples package. You can open this example with Developer to see how the pipeline is mapped between steps.

To run this example, you must have a private key, the associated certificate, and the certificate of the CA who signed it. Some of these credentials are needed by the helper service, sample.smime.helpers:acquireEncryptedMsg, which generates the test message used in this example. You will need to edit the first step in the helper service to specify the location of these files on your system.

When you run this service from Developer, it will prompt you for the following:

<table>
<thead>
<tr>
<th>Input Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recipientsPrivateKeyFile</td>
<td>The name of the file containing the recipient’s private key, for example, d:\certs\myprivatekey.der.</td>
</tr>
<tr>
<td>recipientsCertificateFile</td>
<td>The name of the file containing the certificate belonging to recipientsPrivateKeyFile (the same certificate used to encrypt the message), for example, d:\certs\myCACert.der.</td>
</tr>
</tbody>
</table>
CHAPTER 19  Working with MIME and S/MIME Messages

Flow Service that extracts the content from an encrypted MIME message

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This step acquires an InputStream containing an encrypted multipart MIME message. This example uses a helper service to produce the test message. In a production solution, it is more likely that a MIME message would be passed into the pipeline by a content handler or a back-end system.</td>
</tr>
<tr>
<td>2</td>
<td>This step retrieves the signer’s private key from the file specified in recipientsPrivateKeyFile. Note that the loadAs flag is set to “bytes” to load the file into the pipeline as a byte[].</td>
</tr>
<tr>
<td>3</td>
<td>This step retrieves the signer’s private key from the file specified in recipientsCertificateFile. Note that the loadAs flag is set to “bytes” to load the file into the pipeline as a byte[].</td>
</tr>
<tr>
<td>4</td>
<td>This step takes the InputStream from step 1 and the credentials from steps 2 and 3 and decrypts the message. It produces a MIME object (mimeData) that contains the decrypted message’s constituent elements (header fields and content).</td>
</tr>
<tr>
<td>5</td>
<td>This step extracts each body part from mimeData and appends it to a String list.</td>
</tr>
</tbody>
</table>

Extracting Data from a Signed and Encrypted MIME Message

If your solution receives messages that are signed and/or encrypted, your flow service must test each incoming message and process it appropriately.
Extracting Data from MIME and S/MIME Messages

Example—Extracting Content from a Signed and Encrypted S/MIME Message

The following flow service extracts data from MIME and S/MIME messages. This example resides in sample.smime.extract_SignedAndEncryptedSMime in the WmSamples package. You can open this example with Developer to see how the pipeline is mapped between steps.

To run this example, you must have a private key, the associated certificate, and the certificate of the CA who signed it. Some of these credentials are needed by the helper service, sample.smime.helpers.acquireSignedAndEncryptedMsg, which generates the test message used in this example. You will need to edit the first step in the helper service to specify the location of these files on your system.

When you run this service from Developer, it will prompt you for the following:

<table>
<thead>
<tr>
<th>Input Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recipientsPrivateKeyFile</td>
<td>The name of the file containing the recipient’s private key, for example, d:\certs\myprivatekey.der.</td>
</tr>
<tr>
<td>recipientsCertificateFile</td>
<td>The name of the file containing the certificate that belongs to recipientsPrivateKeyFile (the same certificate used to encrypt the message), for example, d:\certs\myCAcert.der.</td>
</tr>
</tbody>
</table>

Flow Service that extracts the content from a signed and/or encrypted MIME message

```
# Flow Service that extracts the content from a signed and/or encrypted MIME message

# extract_SignedAndEncryptedSMime

acquireSignedAndEncryptedMsg()
createMimeData(
    true, SEQUENCE(
        qFile()
    ))

BRANCH on 'encrypted'(
    true, SEQUENCE(
        qFile()
    )
)

processSignedAndEncryptedData(
    true, SEQUENCE(
        qFile()
    )
)

# BRANCH on 'signed'

BRANCH on 'signed'(
    true, SEQUENCE(
        processSignedData()
    )
)

BRANCH on 'newly'

false, SEQUENCE(
    true, SEQUENCE(
        streamToFile()
    )
)

true, SEQUENCE(
    streamToFile()
)
```

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This step acquires an InputStream containing a signed and encrypted MIME message. This example uses a helper service to produce the test message. In a production solution, it is more likely that a MIME message would be passed into the pipeline by a content handler or a back-end system.</td>
</tr>
</tbody>
</table>
2 This step attempts to create a MIME object from the InputStream produced in step 1.

3 This step tests the `encrypted` flag to see whether the message is encrypted. If it is, it obtains the credentials needed to decrypt the message and passes those credentials and the message to the processEncryptedData service. Note that the `stream` output variable is mapped to the `SMimeEnvStream` input parameter, because the original InputStream from step 1 was emptied by step 2.

Note that if `encrypted` is “false,” execution falls through to step 4.

4 This step tests the `signed` flag to see whether the message is signed. If it is, it passes the message to the processSignedData service. Note that the `stream` output variable is mapped to the `SMimeEnvStream` input parameter, because the original InputStream from step 1 was emptied in step 2. (When a message is decrypted in step 3, the processEncryptedData service produces the `stream` used by this step)

Note that if `signed` is “false,” execution falls through to step 5.

5 This step extracts the data from the `mimeData` object produced by the preceding steps.
Using WebTap

- What Is WebTap? ................................................................. 506
- Creating a WebTap Service ................................................. 508
- Defining WebTap Rules ....................................................... 510
- Creating Before and After Services ................................. 511
- Creating a Customized WebTap Service ............................ 518
- Invoking a WebTap Service from a Browser ....................... 520
What Is WebTap?

WebTap is a facility that acts as an intermediary between a browser-based client and a Web server. You use it to build services that monitor HTTP requests and conditionally execute a prescribed set of actions based on a URL requested by a client. WebTap allows you to build flexible, browser-based applications that combine manual and automated interactions, by letting you filter HTTP requests—letting some requests “pass through” to the target server and intercepting others for special handling.

**Important!** This is the last release of SAP BC Server to support the WebTap feature. It will not be available in future versions of the SAP BC Server.

When a browser makes a request for a WebTap service, all follow-on requests from that client are channeled through the WebTap service. In this capacity, the WebTap service serves as an agent, submitting HTTP requests on the client's behalf. (To the target server, WebTap appears as the client, not the original requestor).

Using WebTap provides the following benefits:

- It allows you to intercept a request and take some action based on the user-supplied data (specifically, the URL and/or name=variable arguments) associated with it.

- It facilitates data-aggregation applications by routing all follow-on requests through a single service, reusing the authentication and cookie context established during the initial request.

- It provides a vehicle through which multiple users can access a protected resource through a common user account.

**Important!** One of the primary differences between a WebTap service and other services is that a WebTap service passes the requested HTML document (modified slightly to ensure that follow-on requests get directed through the WebTap service) back to the client. It does not load and bind the document to convert it to an IData object as regular services do.

Components of a WebTap Application

A basic WebTap application consists of:

- A browser-based client that invokes a WebTap service and passes a single variable (reqUrl) containing the address of the requested resource. (Note that WebTap only works with browser-based clients.) If a request submits arguments in a query string, reqUrl must contain the complete request string (i.e., the URL and the name=value arguments). If the request submits arguments in a posted message (i.e., in the body of the document), you specify the resource's URL in reqUrl and submit the name=value arguments in the body of the message.
A WebTap service (constructed from the RelayHandler class) that optionally has a set of "rules" associated with it defining a URL and/or name=value arguments that, when submitted, will trigger a specified action.

How Does a WebTap Service Work?

The following describes the process that takes place when a WebTap service runs:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WebTap receives the request and compares the user-supplied data (the URL and any submitted name=value pairs) to the &quot;rules&quot; associated with the service.</td>
</tr>
<tr>
<td>2</td>
<td>If the user-supplied data matches the criteria defined in a rule, the server checks to see whether the rule specifies a &quot;before&quot; action (often one or more services). If so, that action is executed.</td>
</tr>
<tr>
<td>3</td>
<td>WebTap submits the request for the specified resource. (Note that if a &quot;before&quot; action was executed, the resource may be different than the one originally requested by the client.)</td>
</tr>
</tbody>
</table>
| 4    | WebTap receives the response and remaps the hypertext links within it (this ensures that all follow-on requests are routed back through the WebTap service). For example, if the response contained the following reference:

```
HREF="http://www.abc.com"
```

WebTap replaces that reference with:

```
HREF="/invoke/folder/Service?reqUrl=MapNum"
```

Where `folder/Service` is the name of the WebTap service and `MapNum` is an alias (dynamically generated by WebTap) for the actual URL assigned to this reference. |
| 5    | WebTap checks to see whether the rule defines an "after" action (usually one or more services) for the request. If there is, that action is executed. |
| 6    | WebTap returns the HTML page (which could have been altered in the previous step) to the client. |

Guidelines for Using WebTap

When creating WebTap services, keep the following points in mind:

- If you want to build a pass-through service (i.e., one that simply relays HTTP requests between a client application and a target server for the purpose of using a common user account or maintaining a single client context), create a WebTap service that contains no rules.
Creating a WebTap Service

A WebTap service is a Java service built around the RelayHandler class. The service calls the “rules engine,” which processes the rules file (if it exists) associated with the service. The rules file defines the criteria (triggers) that cause specified services to execute at runtime.

You build WebTap services using SAP BC Developer. It provides the interface that you use to specify the rules for the service (i.e., it builds the rules file for the service) and generates a source-code stub that comprises the body of the service.

Depending on what you want to do, Developer allows you to start with:

- A standard framework that you can use to create a basic WebTap service based on standard rules (i.e., rules that you can define using the Rules tab in Developer). Start with this framework when you want to create a service that only relays messages between the client and a target server and/or intercepts requests based on URL pattern-matching rules. (Note that you cannot disable the rules engine in this type of service.)

- A custom framework that lets you insert your own code in the service. Use this framework when you want to incorporate custom logic directly in the WebTap service and/or you need a finer level of control than the standard framework provides. For example, you would use the custom framework if you needed to:
  - Execute a block of code before applying the standard rules to a request (e.g., logging a transaction).
  - Apply a complex rule that could not be described using standard WebTap rules (e.g., a condition that requires pattern matching and Boolean logic, for example).
  - Selectively disable the rules engine.

The following procedure describes how to create a WebTap application using either framework.
To create a WebTap Service:

1. On the File menu, click New.
2. Select WebTap Service and then click Next.
3. In the New WebTap Service dialog box, next to Folder, select the folder in which you want to save this service.
4. In the Name field, type a name for the service and click Next.
5. Select the type WebTap service you want to create, and then click Finish.
6. If you want to trigger specific services before and/or after WebTap submits a request to the target service, use the Rules tab to define a rule for each reqUrl value that will trigger an action. See “Defining WebTap Rules” on page 510 if you need procedures for this step.
7. If you selected Custom in step 4, add your code on the Source tab and the Shared tab as needed. For additional information about creating custom code, see “Creating a Customized WebTap Service” on page 518.
8. To test your service, select Run in Browser from the Test menu. Developer prompts you for the value of reqUrl, launches your browser, and invokes the service.

**Important!** Although Developer displays a Settings tab for a WebTap service, only a few of its options should be used with WebTap. (The caching and stateless options must never be used.) The following describes how to use the Settings options with a WebTap service:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>Use this option only when a before service, an after service, or custom code returns a Values object to the client instead of an HTML document. See the rspVals variable in “Creating Before and After Services” on page 511 for information about returning a Values object instead of HTML.</td>
</tr>
<tr>
<td>Runtime</td>
<td>Do not use these options. Make sure they are all disabled.</td>
</tr>
</tbody>
</table>
Defining WebTap Rules

When you create a WebTap service, you can optionally define a set of “rules” specifying actions that you want the service to take if it receives a URL and/or a name=value argument matching certain criteria.

A rule is comprised of:

- A pattern-matching string that describes a URL and/or one or more name=value arguments.
- The name of the service you want WebTap to execute before it submits the request to the target server, and/or...
- The name of the service you want WebTap to execute after it receives a response from the target server.

You may define any number of rules for a WebTap service. If you define multiple rules, WebTap executes all rules matching the submitted request.

The following procedure describes how to define a rule for a WebTap service. To use this procedure, you must first open your WebTap service in Developer and display the Rules tab.

To define a WebTap rule

1. In the Rules tab, click to add a new, unnamed rule to the rule list.
2. Select the new, unnamed rule in the list.
3. In the Rule Name field, type a name for the rule.
4. In the URL Match field, type a pattern-matching string describing the URL that will trigger a before or after service.
   - You can specify a pattern-matching string using standard wildcard characters (see page 573 for allowed wildcards). The following example would match requests for any resource on the www.rubicon server.
     
     Example  http://www.rubicon.*
     
     -OR-
     
     - Use a regular expression. When you use a regular expression, you must enclose the entire pattern-matching string in / characters. The following example would match any URL in which the string “rubicon” appears anywhere.

     Example  /.*rubicon.*/

     See Appendix D, “Regular Expressions” on page 585, for a description of the regular expression syntax.
Creating Before and After Services

If you want to match a particular set of name=value arguments for the service, use the following steps to describe the criteria for each argument that you want to test.

1. Click \(\text{Input For}^{\text{dialog box}}\) to create an empty row in the Input Match list.
2. In the Input For dialog box, type the following information:

   **In this field...** | **Specify...**
   --- | ---
   **Name** | The name of the argument (i.e., the name portion of the name=value pair) that you want to test.
   **Pattern** | A pattern-matching string describing the value that will satisfy this rule.

3. Click OK.
4. Repeat steps 1–3 for each argument that you want to specifically match. Keep the following points in mind when matching on arguments:

   - When you specify multiple arguments in Input Match, the rule is not satisfied unless all of them are true. (A Boolean AND is implied.)
   - If you specify an argument in Input Match, the rule is not satisfied unless that argument appears in the request and it meets the specified criteria.
   - If you do not specify an argument in the Input Match list, any form of the argument will satisfy the rule.

5. If you want this rule, when true, to trigger the execution of a service before the request is submitted, specify the name of that service in the Service Before field.
6. If you want this rule, when true, to trigger the execution of a service after it receives a request from the target server, specify the name of that service in the Service After field.

Creating Before and After Services

*Before* and *after* services are triggered when a WebTap application receives a URL and/or a name=value argument matching a designated pattern. If the pattern (as specified in a WebTap rule) has a before service associated with it, WebTap executes that service before submitting the request to the target server. If the pattern has an after service associated with it, WebTap executes that service after it receives a response from the target server but before returning the response to the client. (See page 507 for an overview of the stages of execution of a WebTap service.)

Any type of service can be used as a before or after service—a flow service, a Java service, a C/C++ service and so forth. In most cases, you will want the service to act upon information available to the WebTap service at run time. To do this, you must assign the `pub.webtap:triggerSpec` specification to your service (this specification resides in the WmPublic package). It defines the variables that WebTap makes available to before and
after services. You use these variables to read and/or modify data that is passed between
the client and the target server and to control the behavior of WebTap at run time.

For example, you can change the reqUrl variable in a before service to reroute the client’s
original request to a different resource. (See “A Simple Rerouting Example” on page 516
for an example of this technique.) You can also use the rspVals or rspTxt control variable to
“short-circuit” the WebTap process and return a predefined set of outputs to the client,
instead of passing the request to the target server.

The following table describes the specification of a WebTap service. Note that it is made
up of two records: WebtapData and WebtapControl. WebtapData holds the data that
passes between the client and the target server. WebtapControl contains a set of control
variables that you use to direct the run time behavior of the WebTap service.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebtapData</td>
<td>A Values object containing the inputs to the service, including reqUrl and data posted by the requestor or passed via a query string. This object contains one element for each argument submitted with the request.</td>
</tr>
<tr>
<td>WebtapControl</td>
<td>Contains the following elements that you use to control the run-time behavior of WebTap.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reqUrl</td>
<td>Specifies the URL of the requested resource (as specified in pub.webtap:getPage). You can modify this value in a before service to redirect a request.</td>
</tr>
<tr>
<td>reqVals</td>
<td>Specifies the set of substitute inputs to use when redirecting a request to a different URL.</td>
</tr>
<tr>
<td>rspUrl</td>
<td>Set on response (read-only) if the request was redirected to indicate the URL of the returned page.</td>
</tr>
<tr>
<td>rspVals</td>
<td>Contains a Values object to return to the client. Setting this value interrupts the normal WebTap sequence and returns this Values object instead of the requested page (if you set rspVals in a before service, the client’s original request is not submitted to the target server by WebTap).</td>
</tr>
<tr>
<td>rspTemplate</td>
<td>Specifies the name of the template to use to format the contents of rspVals. (This setting overrides the template setting associated with the service.)</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>rspTxt</strong></td>
<td>Contains an HTML document to return to the client. Setting this value interrupts the normal WebTap sequence and returns this document instead of the requested page (if you set <code>rspTxt</code> in a before service, the client's original request will not be submitted to the target server by WebTap).</td>
</tr>
<tr>
<td><strong>authUser</strong></td>
<td>Specifies the HTTP authentication user name that the service uses when requesting a protected resource.</td>
</tr>
<tr>
<td><strong>authPwd</strong></td>
<td>Specifies the HTTP authentication password that the service uses when requesting a protected resource.</td>
</tr>
<tr>
<td><strong>authUrl</strong></td>
<td>Specifies the HTTP authentication URL that the service uses when requesting a protected resource.</td>
</tr>
<tr>
<td><strong>mapImages</strong></td>
<td>Contains the string <code>true</code> or <code>false</code>, specifying whether WebTap should map URLs associated with images in the returned document. (Set to <code>false</code> if images are not password-protected). Default is <code>true</code>.</td>
</tr>
<tr>
<td><strong>mapPage</strong></td>
<td>Contains the string <code>true</code> or <code>false</code>, specifying whether WebTap should map URLs in the returned document. You may set this element to <code>false</code> to exit a WebTap service. Default is <code>true</code>.</td>
</tr>
<tr>
<td><strong>mapJavascript</strong></td>
<td>Contains the string <code>true</code> or <code>false</code>, specifying whether WebTap should map URLs embedded in JavaScript in the returned document. (Note that this mapping handles many common cases of dynamic JavaScript URL creation; however, it does not handle <code>document.write()</code> and <code>eval()</code> blocks that involve URL definition.) Default is <code>false</code>.</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td><em>showEmbed</em></td>
<td>Contains the string true or false, specifying whether WebTap will transform the URLs associated with <code>&lt;EMBED&gt;</code> elements.</td>
</tr>
<tr>
<td></td>
<td>- Set to true if you want WebTap to remap these URLs to fully qualified Web addresses. This setting enables the requestor’s browser to retrieve the embedded resource from the page returned by your WebTap service.</td>
</tr>
<tr>
<td></td>
<td>- Set to false if you want WebTap to strip attributes from <code>&lt;EMBED&gt;</code> elements, which will prevent embedded resources from appearing in the page returned to the requestor.</td>
</tr>
<tr>
<td></td>
<td>Default is false.</td>
</tr>
<tr>
<td><em>showApplet</em></td>
<td>Contains the string true or false, specifying whether WebTap will transform URLs associated with <code>&lt;APPLET&gt;</code> elements.</td>
</tr>
<tr>
<td></td>
<td>- Set to true if you want WebTap to remap these URLs to fully qualified Web addresses. This setting enables the requestor’s browser to retrieve the applet from the page returned by your WebTap service.</td>
</tr>
<tr>
<td></td>
<td>- Set to false if you want WebTap to strip attributes from <code>&lt;APPLET&gt;</code> elements, which will prevent applets from appearing in the page returned to the requestor.</td>
</tr>
<tr>
<td></td>
<td>Default is false.</td>
</tr>
</tbody>
</table>
### Variable Name | Description
--- | ---
**showObject** | Contains the string `true` or `false`, specifying whether WebTap will transform URLs associated with `<OBJECT>` element.
  - Set to `true` if you want WebTap to remap these URLs to fully qualified Web addresses. This setting enables the requestor’s browser to retrieve the object from the page returned by your WebTap service.
  - Set to `false` if you want WebTap to strip attributes from `<OBJECT>` elements, which will prevent objects from appearing in the page returned to the requestor.
  
  Default is `false`.

**matchSSL** | Contains the string `true` or `false`, specifying whether WebTap should map https-based URLs to https-based aliases. Set `matchSSL` to `true` if you want to use an SSL connection between the client and SAP BC Server when linking to a site whose URL specifies the https protocol.

  Default is `false`.

**node** | Contains a parsed representation of the HTML document retrieved by WebTap (before URL mapping occurs). You can use this document as input to any service that accepts a node as input (e.g., `queryDocument`).

---

*Note:* Before using this option, verify with your server administrator that an https port is configured on your SAP BC Server.
A Simple Rerouting Example

The following general procedure describes the steps that you take to build a WebTap application that intercepts requests for a particular URL and routes them to another URL.

### Debugging a WebTap Service

To debug a WebTap service whose rules are not performing as you expect, enable the trace flag in the Java code for that service. When you enable this flag, WebTap records the service’s variable settings in the error log on SAP BC Server.

To trace a WebTap service, perform the following general steps:
Creating Before and After Services

Step | Description
--- | ---
1 | Open the WebTap service in the Developer and add the following line of code to the **Source** tab.

```java
handler.setTrace(true);
```

2 | Execute the service via **Test > Run In Browser**.

3 | View the console output or examine the contents of the following log file on SAP BC Server to see the results from the trace:

```plaintext
<sapbc>\server\logs\server.log
```

It will show the values of variables that the service set, and will look similar to the following:

**Output from the Trace of a WebTap Service**

```plaintext
> 000265 WEBTAP9999C --- START tracePipeline [Thu Mar 02 18:25:25 EST 2000]---
> 000266 WEBTAP9999C reqUrl = http://www.compusapc.com/Images/aol_sign_up.gif
> 000267 WEBTAP9999C reqUrlList = [Ljava.lang.String;@210eb7
> 000268 WEBTAP9999C rspUrl = null
> 000269 WEBTAP9999C --- END tracePipeline ---
```
Creating a Customized WebTap Service

If you want to build a WebTap service that executes a standard block of code before applying any rules to the request, or a WebTap service that requires trigger conditions that cannot be described using WebTap’s pattern-matching rules, you can build a custom WebTap service.

When you choose to start with the custom framework, Developer gives you a source-code stub that looks as follows. (Shaded areas show where the rules engine is called. Rules associated with the service are evaluated at these points, and the appropriate before and/or after services are executed.)

```java
RelayIf WebTapCustom__ = new RelayIf()
{
    RelayHandler handler = new RelayHandler(this);
    NSName svcName = Service.getServiceEntry().getNSName();
    public Values relay(Session session, Values in)
    {
        return handler.relay(session, in);
    }
    // Request Callback - Add your request triggers here
    public boolean relayRequest(Session session, Values in, String reqUrl)
    throws Exception
    {
        // your triggers
        // rules engine (optional)
        if (handler.processRequestRules(svcName) == RelayHandler.DONT_CONTINUE)
            return false;
        return true;
    }
    // Response Callback - Add your response triggers here
    public boolean relayResponse(Session session, String doctxt, String rspUrl)
    throws Exception
    {
        // your triggers
        // rules engine (optional)
        if (handler.processResponseRules(svcName) == RelayHandler.DONT_CONTINUE)
            return false;
        return true;
    }
};
out = WebTapCustom__.relay(Service.getSession(), in);
```

Add custom code and “before” tests here.

Add custom code and “after” tests here.
Example

The following code segment shows a WebTap service that logs an incoming request before passing it to the rules engine.

```java
RelayIf logApp__ = new RelayIf()
{
    RelayHandler handler = new RelayHandler(this);
    NSName svcName = Service.getServiceEntry().getNSName();
    public Values relay(Session session, Values in)
    {
        return handler.relay(session, in);
    }
    // Request Callback - Add your request triggers here
    public boolean relayRequest(Session session, Values in, String reqUrl)
    throws Exception
    {
        logRequest(svcName, handler, reqUrl);
        // your triggers
        // rules engine (optional)
        if (handler.processRequestRules(svcName) == RelayHandler.DONT_CONTINUE)
            return false;
        return true;
    }
    .
    .
    .
}
```

You can also use the custom framework to define your own set of rules (i.e., test for certain conditions in the URL or user data) by inserting code at the appropriate trigger points.
Example

The following code fragment shows a WebTap service that contains a custom trigger (shaded) that executes a special block of code if `reqUrl` contains a specified string.

```java
RelayIf WebTapCustom__ = new RelayIf()
{
    RelayHandler handler = new RelayHandler(this);
    NSName svcName = Service.getServiceEntry().getNSName();
    .
    .
    // Response Callback - Add your response triggers here
    public boolean relayResponse(Session session, String doctxt, String rspUrl)
    {
        // checkout trigger
        if (reqUrl.indexOf("secure_checkout.asp?chkout=yes") > 0)
        {
            // intercept request and return contents to caller
            try{
                // invoke flow to extract contents and set alternate response
                handler.setAltResponse(Service.doInvoke("sample.webtap","getcart", session,in),Compuse_retrieve");
            } catch (Exception e)   {
                throw(e);
            }
            // return alternate response to caller
            return false;
        }
        // rules engine (optional)
        if (handler.processResponseRules(svcName) == RelayHandler.DONT_CONTINUE)
        {
            return false;
        }
        return true;
    }

    // checkout trigger
    if (reqUrl.indexOf("secure_checkout.asp?chkout=yes") > 0)
    {
        // intercept request and return contents to caller
        try{
            // invoke flow to extract contents and set alternate response
            handler.setAltResponse(Service.doInvoke("sample.webtap","getcart", session,in),Compuse_retrieve");
        } catch (Exception e)   {
            throw(e);
        }
        // return alternate response to caller
        return false;
    }
    // rules engine (optional)
    if (handler.processResponseRules(svcName) == RelayHandler.DONT_CONTINUE)
    {
        return false;
    }
    return true;
}

out = WebTapCustom__.relay(Service.getSession(), in);
```

For additional information about the WebTap API, see the RelayHandler class in the SAP BC Java API Reference.

Invoking a WebTap Service from a Browser

You use the following URL to invoke a WebTap application from a browser.

```
http://servername/invoke/Folder/Service?reqUrl=RequestedUrl
```

Where:

`servername` is the name of the SAP BC Server on which the service resides.
Folder is the name of the folder in which the service resides.

Service is the name of the WebTap service.

RequestedUrl is the URL of the requested resource. This part of the string must be URL-encoded!

Example

Subscribing to Events

- The Event Manager .......................................................... 524
- Managing Event Subscriptions ......................................... 526
- Building an Event Handler ................................................. 533
- Working with Alarm Events .............................................. 535
- Working with Audit Events .............................................. 536
- Working with Exception Events ....................................... 540
- Working with Guaranteed Delivery Events .......................... 542
- Working with Port Status Events ..................................... 545
- Working with Replication Events ..................................... 546
- Working with Session Events .......................................... 546
- Working with Stat Events ............................................... 549
- Working with Transaction Events .................................... 550
The Event Manager

The Event Manager monitors the server for events and invokes event handlers when those events occur. An event is a specific action that the Event Manager recognizes and an event handler can react to. An event handler is a service that you write to perform some action when a particular event occurs.

Currently, the Event Manager recognizes the following types of events:

- **Alarm events** occur when SAP BC Server throws an exception regarding the status or health of the server. The server generates alarm events when a user cannot log on to the server, a port cannot be started, a user is denied access to a port, an error occurs in Cluster Manager, or a service cannot execute because of errors. Subscribe to alarm events to trigger event handlers that perform specific actions such as notifying administrators about port access exceptions and service failures, or sending information to a console when a port cannot be started.

- **Audit events** occur when a service begins and when it ends. Anytime a service executes, it produces two audit events—one when it starts and another when it finishes. Subscribe to audit events to trigger specific actions when a particular service or class of service executes.

- **Exception events** occur every time a service throws an exception. Subscribe to exception events to trigger specific actions when a particular service or class of service fails.

- **Guaranteed Delivery events** occur when a client uses guaranteed delivery to invoke a service on a SAP BC Server and when the server returns the service results to the requesting client. There are two types of guaranteed delivery events—GD Start and GD End. Subscribe to GD Start and GD End events to trigger event handlers that perform actions such as logging guaranteed delivery transactions to a file or sending notification.

- **Port Status events** occur each time SAP BC Server updates the server statistics. The port status event provides information about the status of all the ports configured on SAP BC Server. Subscribe to port status events to trigger event handlers that perform actions such as sending port status data to a network monitoring system or writing port status data to a log file.

- **Replication events** occur when the pub.replicator:generateReplicationEvent service executes. Subscribe to replication events to trigger event handlers that perform actions such as notifying package subscribers when a package is published and maintaining a log of pulled or distributed packages.

- **Session events** occur when a client starts or ends a session on SAP BC Server or when SAP BC Server terminates an inactive session. There are three types of session events—session start, session end, and session expire. Subscribe to session events to trigger event handlers that perform actions such as maintaining your own log files.

- **Stat events** occur each time SAP BC Server updates the statistics log (stats.log). Subscribe to stat events to trigger event handlers that perform actions such as...
maintaining your own log file or sending server statistics to a network monitoring system.

- **Transaction events** occur when a SAP BC Server begins and finishes processing a guaranteed delivery transaction. There are two types of transaction events—Tx Start and Tx End. Subscribe to transaction events to trigger event handlers that perform specific actions (such as sending notification or logging information) when a particular guaranteed delivery transaction begins or finishes processing.

## What Are Event Handlers?

An *event handler* is a service that you write to perform some action when a particular event occurs. (An event handler can be any type of service—a flow service or a Java service.) Event handlers *subscribe* to the events that they want to be notified of. For example, if you wanted an event handler to execute when a particular service throws an exception, you subscribe the event handler to the exception event for that service.

## What Happens When an Event Occurs?

When an event occurs, the Event Manager automatically invokes all event handlers that subscribe to the event. The event handlers receive an input object containing run-time information. The exact content of this input object varies depending on the type of event that occurred and, for audit events, the run-time properties set on both SAP BC Server and the service that generated the event.

Once an event handler is invoked, its execution is completely asynchronous of the event that triggered it. If the execution of a service triggers an event handler (as with audit and exception events), the execution path of the triggering service is not blocked or altered in any way (in fact, it is never even aware that it triggered an event handler).

Other points to keep in mind about events and event handlers:

- An event can have more than one subscriber, which means that a single event might trigger several event handlers.

- If an event triggers more than one event handler, all the event handlers execute simultaneously. They *do not* execute serially and they are not invoked in any particular order. (If you have a series of actions that must execute in a specific sequence, you should encapsulate the entire sequence within a single event handler.)

- An event handler can subscribe to more than one event.

- When event handlers run, they do not generate audit events.

- If an event handler throws an exception, it generates an exception event. This is true for all event handlers *but* exception event handlers. When an exception event handler throws an exception, it does not generate an exception event.
Managing Event Subscriptions

You can use the Event Manager in Developer to manage all of your event subscriptions. The Event Manager can perform the following tasks:

- Subscribe event handlers to events
- View or edit event subscriptions
- Suspend event subscriptions
- Delete event subscriptions

The following sections contain more information about each of these tasks.

**Note:** You can also use built-in services to add, modify, and delete event subscriptions. These services are located in the pub.event folder. For more information about built-in services, see the SAP BC Built-In Services Guide.

**Subscribing to an Event**

You can use the Event Manager in Developer to subscribe to an event on the current server. This action registers the event handler with the Event Manager and specifies which events will invoke it. To access the Event Manager, select Edit > Event Manager.

Use the Event Manager in Developer to subscribe to events.
Use the following procedure to subscribe to an event on the current server. To perform this procedure, you must have already:

- Identified the event type you want to subscribe to
- Identified the service or services that generate an event you want to subscribe to (if you want to subscribe to an audit event, exception event, or GD Start event).
- Written the event handler that will execute when the identified event occurs

To subscribe to an event on the current server

1. On the Edit menu, select Event Manager.
2. In the Event Manager dialog box, in the View event subscribers for list, select the event type to which you want to subscribe.
3. Click to add a new subscriber.
4. In the Enter Input Values dialog box, complete the following fields:
In this field... Specify...

Service  The fully qualified name of the event handler that will subscribe to the event (that is, the service that will execute when the event occurs).

Example:  sgxorders.Authorization:LogAuthTrans

Filter  A pattern string to further limit the events this event handler subscribes to. Filters will vary depending on the event type you are subscribing to.

For example, if you are subscribing to an audit or exception event, create a filter to specify the names of services whose events this event handler subscribes to (that is, the services that, when executed, will trigger the event handler specified in Service).

You may use the characters *, ?, and ^ as wildcards:

* replaces any number of characters (can be used anywhere in a pattern).

? replaces one character (can be used anywhere in a pattern).

^negates the specified characters (can only be used at the beginning of a pattern).

Example: ‘^credit’ matches everything but patterns containing ‘credit.’

The pattern string is case sensitive.

For more information about creating event filters, see “Creating Event Filters” on page 529.

Comment  An optional descriptive comment about this subscription.

Enabled  Whether the subscription is active or inactive. Set to true to activate the subscription. Set to false to deactivate the subscription. (This allows you to temporarily suspend a subscription without deleting it.)

Click OK. Subscriptions take effect immediately.

Note: SAP BC Server saves information for event types and event subscriptions in the eventcfg.bin file. This file is generated the first time you start SAP BC Server and is located in the following directory: <sapbc>\server\config. Copy this file from one SAP BC Server to another to duplicate event subscriptions across servers.
Creating Event Filters

Event filters allow you to be very selective about which events you subscribe to. Event filters limit the events for an event type that trigger an event handler. By using event filters, you can subscribe an event handler to only those events generated by a particular service, package, user, or port. For example, you might want an event handler to be triggered only when a specific service generates an audit event. Or, you might want an event handler to be triggered only when a specific user logs on to SAP BC Server.

The following table identifies the information that you can filter on for each event type. Notice that you cannot create a filter for some event types. For these event types, every generated event triggers the event handlers subscribed to it.

<table>
<thead>
<tr>
<th>For this event type...</th>
<th>You create a filter for...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm Event</td>
<td>The message generated by the alarm event. Create a filter that specifies some of the text of the message. The event handler with this filter will process all alarm events containing the specified text.</td>
</tr>
<tr>
<td></td>
<td>The following filter specifies that any alarm events that generate a message containing the word “port” will trigger the event handler:</td>
</tr>
<tr>
<td></td>
<td><em>port</em></td>
</tr>
<tr>
<td>Audit Event</td>
<td>The fully qualified name of the service that generates the audit event. Create a filter to specify the services whose audit events you want to trigger the event handler.</td>
</tr>
<tr>
<td></td>
<td>The following filter specifies that the service sgxorders.Authorization:creditAuth will trigger the event handler:</td>
</tr>
<tr>
<td></td>
<td>sgxorders.Authorization:creditAuth</td>
</tr>
<tr>
<td>Exception Event</td>
<td>The fully qualified name of the service that generates the exception event. Create a filter to specify the services whose exception events you want to trigger the event handler.</td>
</tr>
<tr>
<td></td>
<td>The following filter specifies that all services that start with the word “credit” and belong to any folder will trigger the event handler:</td>
</tr>
<tr>
<td></td>
<td><em>:credit</em></td>
</tr>
<tr>
<td>GD End Event</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>The filter for all GD End events will be the following:</td>
</tr>
<tr>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

Important! *?, and ^ are the wild-card characters allowed in an event filter. All other characters in the pattern string are treated as literals. Pattern strings are case sensitive.
### GD Start Event

The fully qualified name of the service that is being invoked using guaranteed delivery. Create a filter to specify the services that when invoked using guaranteed delivery will trigger the event handler.

The following pattern string specifies that all services that start with the word “sendPO” and belong to any folder will trigger the event handler:

```
*:sendPO*
```

### Port Status Event

N/A

The filter for all port status events will be the following:

```
*
```

### Replication Event

The name of the package being replicated. Create a filter to specify the packages that when replicated will trigger the event handler.

The following filter specifies that a replication event involving the package named “AcmePartnerPkg,” will trigger the event handler:

```
AcmePartnerPkg
```

### Session End Event

N/A

The filter for all session end events will be the following:

```
*
```

### Session Expire Event

N/A

The filter for all session expire events will be the following:

```
*
```

### Session Start Event

The user name for the user starting the session on SAP BC Server or the groups to which the user belongs. Create a filter to specify which users or which user groups trigger an event handler when they start a session on the server.

The following filter specifies that a session start event generated by a user in the “Administrators” group will trigger the event handler.

```
*Administrators*
```

### Stat Event

N/A

The filter for all stat events will be the following:

```
*
```

---

For this event type... You create a filter for...

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD Start Event</td>
<td>The fully qualified name of the service that is being invoked using guaranteed delivery. Create a filter to specify the services that when invoked using guaranteed delivery will trigger the event handler. The following pattern string specifies that all services that start with the word “sendPO” and belong to any folder will trigger the event handler: <code>*:sendPO*</code></td>
</tr>
<tr>
<td>Port Status Event</td>
<td>N/A</td>
</tr>
<tr>
<td>Replication Event</td>
<td>The name of the package being replicated. Create a filter to specify the packages that when replicated will trigger the event handler. The following filter specifies that a replication event involving the package named “AcmePartnerPkg,” will trigger the event handler: <code>AcmePartnerPkg</code></td>
</tr>
<tr>
<td>Session End Event</td>
<td>N/A</td>
</tr>
<tr>
<td>Session Expire Event</td>
<td>N/A</td>
</tr>
<tr>
<td>Session Start Event</td>
<td>The user name for the user starting the session on SAP BC Server or the groups to which the user belongs. Create a filter to specify which users or which user groups trigger an event handler when they start a session on the server. The following filter specifies that a session start event generated by a user in the “Administrators” group will trigger the event handler: <code>*Administrators*</code></td>
</tr>
<tr>
<td>Stat Event</td>
<td>N/A</td>
</tr>
</tbody>
</table>

---

For this event type... You create a filter for...
Creating Event Filters for Services

When you can create a filter for a service name, you can be very selective about which service’s events you subscribe to. You can use regular expressions to create event filters for service names. The following examples show ways you can use regular expressions as event filters to specify an event that a particular service generates.

For this event type... You create a filter for...

Tx End Event N/A
The filter for all Tx End events will be the following:
*  
Tx Start Event N/A
The filter for all Tx Start events will be the following:
*  

Creating Event Filters for Services

Viewing and Editing Event Subscriptions

Use the following procedure to view or edit an event subscription on the current server.

1. On the Edit menu, select Event Manager.
2. In the View event subscribers for list, select the event type for which you want to view subscriptions.

To view or edit an event subscription on the current server
3. Click the subscription you want to edit, and then click OK.

4. Modify the fields in the Enter Input Values dialog box as needed and then click OK.

5. Repeat steps 2–4 for each subscription you want to view or edit.

6. Click OK when you finish viewing or editing event subscriptions. Your changes take effect immediately.

### Suspending Event Subscriptions

You can suspend an event subscription. By suspending an event subscription, you temporarily stop the execution of the event handler without deleting or removing the event handler. While the event subscription is suspended, the Event Manager does not invoke the associated event handler when the server generates the event to which it is subscribed. You can resume an event subscription at any time.

**To suspend an event subscription**

1. On the Edit menu, select Event Manager.

2. In the Event Manager dialog box, in the View event subscribers for list, select the event type for which you want to suspend a subscription.

3. Click the subscription you want to edit, and then click OK.

4. In the Enter Input Values dialog box, in the Enabled list, select false.

5. Repeat steps 2–4 for each event subscription you want to suspend.

6. Click OK when you finish suspending event subscriptions. Your changes take effect immediately.

### Deleting an Event Subscription

Use the following procedure to delete an event subscription from the Event Manager.

**To delete an event subscription**

1. On the Edit menu, select Event Manager.

2. In the Event Manager dialog box, in the View event subscribers for list, select the event type for which you want to delete a subscription.

3. Click the subscription you want to delete, and then click .

4. Repeat steps 2 and 3 for each subscription you want to delete.

5. Click OK when you finish deleting events. Your changes take effect immediately.
Building an Event Handler

Building an event handler is a process that involves the following basic stages:

- **Stage 1**  
  Creating an empty service. During this stage, you create the empty service that you want to use as an event handler.

- **Stage 2**  
  Declaring the input and output. During this stage, you declare the input and output parameters for the event handler by selecting the specification or record for the event type in pub.event. The specification and record indicate the run-time data that will be contained in the IData object passed to the event handler.

- **Stage 3**  
  Insert logic, code, or services. During this stage, you insert the logic, code, or services to perform the action you want the event handler to take when the event occurs. If you are building a flow service, make sure to map data between services and the pipeline.

- **Stage 4**  
  Testing and debugging the service. During this stage, you use the testing and debugging tools available in Developer to make sure the event handler works properly.

- **Stage 5**  
  Subscribing to the event. During this stage, you use the Event Manager to subscribe the event handler to the event. This action registers the event handler with the Event Manager and specifies which events will invoke it. You can create filters to be more selective about which events you subscribe to.

Sample Event Handler

Following are instructions for building an event handler named processLogon. The processLogon event handler sends a notification to a specified email address (such as the Server Administrator) when anyone in the Administrators group logs in to SAP BC Server.

- **Stage 1**  
  To create the event handler
  - Create an empty flow service and name it processLogon.

- **Stage 2**  
  To assign input and output parameters to the event handler
  1. On the Input/Output tab, next to the Specification Reference field, click 📄.
  2. In the Select dialog box, navigate to and select the pub.event:sessionStart specification.
  3. Click OK.
CHAPTER 21 Subscribing to Events

Stage 3  To insert services into the event handler

1. Click the Flow tab.
2. On the Flow Editor toolbar, click ➔ and select Browse.
3. In the Browse dialog box, navigate to and select the pub.client:smtp service. Click OK.
4. In the Pipeline Editor, use the Set Value modifier to assign values to the following variables in Service In:

   For this field... Specify...
   to The email address for the person to send event notification to.
   subject %userid% logged in
   The subject of the email message will contain the user ID of the person who logged in to SAP BC Server as a member of the Administrators group.
   mailhost The network name of your mailhost (e.g., mail@mycompany.com).
   body Administrators user %userid% logged into session %sessionName% with session ID %ssnid%
   The body of the email message will contain the user name, session name, and session ID for the person who logged in to SAP BC Server as an Administrator.

5. Click ☐ to save the service.

Stage 4  To test and debug the event handler

- Use the testing and debugging tools in Developer (such as Test ➔ Run) to test and debug the service. For more information about these tools, see “Testing and Debugging Services” on page 247.

Stage 5  To subscribe the event handler to the event

1. On the Edit menu, select Event Manager.
2. In the Event Manager dialog box, in the View event subscribers for list, select Session Start Event.
3. Click ☐ to add a new event subscription.
4. In the Enter Input Values dialog box, complete the following fields:
Working with Alarm Events

An alarm event occurs when SAP BC Server generates a message related to the status of the server. An alarm event can be generated for the following reasons:

- A client experiences a logon failure or is denied access to SAP BC Server. A client cannot log on because of “invalid credentials.”
- Errors occur in the Cluster Manager. The inability to add a port to a cluster can cause errors in Cluster Manager.
- A user tries to access a port and is denied access to the port. (This can happen when a user tries to execute a service not allowed on the port.) This type of alarm event is sometimes called a port access exception.
- A WebTap service is missing a required URL or alias. This is called a relay exception.
- A port cannot be started. The most common reason a port cannot start is that the port is being accessed by another application.
- A service cannot be loaded or executed due to setup errors. For a flow service, a possible error is a missing XML metafile. For a Java service, possible errors include a missing class file or method.

You can use alarm events to trigger event handlers that execute when the server generates messages related to the status of the server. For example, you might want to create an event handler that notifies the administrator when a user is denied access to the server or to a port, when a service fails to load or execute, or when a port does not start. You can also create event handlers to send data to a network monitoring system.

5 Click OK. Subscriptions take effect immediately.
Building Handlers for Alarm Events

When the Event Manager invokes an event handler for an alarm event, the event handler receives an IData object containing the following run-time data. (A specification and record for these parameters are provided in the pub.event folder.)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>A String containing the date and time that the alarm event occurred, in the format yyyy/MM/dd HH:mm:ss.SS</td>
</tr>
<tr>
<td>service</td>
<td>A String containing the fully qualified name of the service that generated the event. A service can generate an alarm event when a client invokes a service that accesses information or a service on a remote server. If the client is not a member of an allowed group for the port on the remote server, the service will generate an alarm event.</td>
</tr>
<tr>
<td>sessionID</td>
<td>A String containing the identification number for the session during which the alarm event was generated. Some alarm events are not generated during sessions. In these cases, the sessionID variable will not contain a value.</td>
</tr>
<tr>
<td>msg</td>
<td>A String containing the error message from the alarm event.</td>
</tr>
</tbody>
</table>

Tip! When you subscribe an event handler to an alarm event, you can create a filter for the msg field to be more selective about the alarm events that trigger the event handler.

Working with Audit Events

By default, when a service executes, two audit events are produced. The first occurs just before a service starts executing, and the second occurs immediately after a service finishes processing. Event handlers that subscribe to audit events are triggered at both points. For example, if your event handler subscribes to the audit events for a service called SGorders:postPO, it will be called twice each time SGorders:postPO executes.

Tip! Audit events are generated regardless of whether a service executes successfully or not. If a service throws an exception, it generates both an exception event and audit events.

Audit events are generated by all services that execute, including nested services. For example, a flow service that has two child services will produce six audit events: two from the flow service itself, and two each from its children. The same is true for coded services (such as C services or Java services).
Setting Audit Levels

Services have an **Audit Level** property, which determines whether they generate an audit event at run time, and if so, how much data they pass to the event handler. You specify the **Audit Level** property on the service’s **Settings** tab.

**Settings tab**

Specify a service’s Audit Level with the Settings tab.
A service can have one of the following Audit Level values:

<table>
<thead>
<tr>
<th>Audit Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>The service does not generate audit events.</td>
</tr>
<tr>
<td>brief</td>
<td>The service generates audit events and passes a compact set of input data to the event handler. This option does not pass a copy of the pipeline to the event handler. This is the default setting.</td>
</tr>
<tr>
<td>verbose</td>
<td>The service generates audit events and passes the complete set of input data to the event handler. This option passes a copy of the pipeline to the event handler.</td>
</tr>
</tbody>
</table>

Additionally, the generation of audit events is contingent on the value of the server’s watt.server.auditLog property. This property globally governs whether audit events are generated. By default, watt.server.auditLog is set to persvc, which means that audit events are generated based on the service’s Audit Level setting. However, you can globally override the individual service Audit Level settings with watt.server.auditLog, as described in the following table.

<table>
<thead>
<tr>
<th>watt.server.auditLog Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>persvc</td>
<td>Services generate audit events according to their Audit Level setting. This is the default setting.</td>
</tr>
<tr>
<td>off</td>
<td>Services do not generate audit events.</td>
</tr>
<tr>
<td>brief</td>
<td>All services generate audit events and pass compact sets of input data to the event handler unless their Audit Level is set to verbose; in which case, they pass complete sets of input data.</td>
</tr>
<tr>
<td>verbose</td>
<td>All services generate audit events and pass complete sets of input data to the event handler. This option should only be used when circumstances truly demand it. The overhead of copying the pipeline before and after every service can degrade server performance.</td>
</tr>
</tbody>
</table>

⚠️ **Important!** The settings in watt.server.auditLog override the individual service Audit Level settings.

### Building Handlers for Audit Events

When the Event Manager invokes an event handler for an audit event, the event handler receives an IData object that contains the following run-time data. (A specification and a record for the input data are provided in the pub.event folder.)
The audit event handlers that you build can make use of these inputs as they need to. Keep in mind that an audit event handler is called twice each time a service runs—once when the service starts and again when it ends. Your event handler can check the value of the `result` parameter to determine whether it is processing an audit event from before or after the service executed.

Also, if your event handler needs data from the triggering service’s pipeline, make sure that service’s Audit Level is set to `verbose`. Otherwise, the `pipeline` element won’t be included in the input object that is passed to your event handler.

Tip! When you subscribe an event handler to an audit event, you can create a filter for the `service` field to specify the services whose audit events you want to subscribe to—that is, you can specify which services’ audit events trigger the event handler.
CHAPTER 21 Subscribing to Events

Default Audit Event Handler

By default, SAP BC Server is shipped with one audit event handler: pub.event.audit.logToFile. This event handler subscribes to all audit events. It is the mechanism that the server uses to log information to the \<sapbc\>\server\logs\audit.log file. Unless you explicitly want to eliminate or replace this server behavior, do not delete or modify this subscription.

For more information about the audit.log file, see the SAP BC Administration Guide.

Working with Exception Events

An exception event occurs when a service throws an exception (including when a flow service “exits on failure”). You can use exception events to trigger some prescribed action—such as notifying an administrator—when a particular service fails.

Note: Keep in mind that event handlers are processed independent of the services that trigger them. They are completely separate, asynchronous processes. Event handlers are not designed to replace the error handling and/or error recovery procedures that you would normally include in your service.

Exception events are always generated when an exception occurs. They are not subject to the Audit Level settings or the watt.server.auditLog property in the same way audit events are. (For information about these settings, see “Setting Audit Levels” on page 537.)

Services that generate exception events also generate audit events when they end (assuming that audit events are not suppressed by the current Audit Level or watt.server.auditLog settings).

If a nested service throws an exception, an exception event is generated by each service in the call stack. For example, if service A1 calls service B1, and B1 throws an exception, both B1 and A1 generate exception events (in that order).

Building Handlers for Exception Events

When the Event Manager invokes an event handler for an exception event, the event handler receives an IData object containing the following run-time data. (A Specification and a record for this signature reside in the pub.event folder.)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>A String containing the date and time that the event occurred, in the format yyyy/MM/dd HH:mm:ss.SS</td>
</tr>
<tr>
<td>error</td>
<td>A String containing the error message from the exception.</td>
</tr>
<tr>
<td>errorType</td>
<td>A String containing the exception type that was thrown.</td>
</tr>
</tbody>
</table>
Working with Exception Events

**Default Exception Event Handler**

By default, SAP BC Server is shipped with one exception event handler: `pub.event.exception:logToFile`. This event handler subscribes to all exception events. It is the mechanism that the server uses to log exceptions to the `<sapbc>\server\logs\error.log` file and to issue emails as specified by the server’s SMTP settings. Unless you explicitly want to eliminate or replace this server behavior, do not delete or modify this subscription.

For more information about the error.log file, see the *SAP BC Administration Guide*. 

---

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorDump</td>
<td>A String containing more detailed information about the exception (if the exception object contains dump information).</td>
</tr>
<tr>
<td>service</td>
<td>A String containing the fully qualified name of the service that generated the event.</td>
</tr>
<tr>
<td>user</td>
<td>A String containing the user that requested the service that generated this event.</td>
</tr>
<tr>
<td>callStack</td>
<td>A Record List containing the items in the call stack. See the <code>pub.event:callStackItem</code> record for the definition of the records in callStack.</td>
</tr>
<tr>
<td>pipeline</td>
<td>A copy of the state of the pipeline at the point when the exception occurred.</td>
</tr>
<tr>
<td>ssnid</td>
<td>A String containing the identification number of the session during which the exception occurred.</td>
</tr>
<tr>
<td>threadID</td>
<td>A String identifying the thread that invoked the service.</td>
</tr>
</tbody>
</table>

**Tip!** When you subscribe an event handler to an exception event, you can create a filter for the service field to specify the services whose exception events you want to subscribe to—that is, you can specify which services’ exception events trigger the event handler.
Working with Guaranteed Delivery Events

A guaranteed delivery event occurs when a client uses guaranteed delivery to invoke a service on a remote SAP BC Server, and when the server returns the service results to the requesting client. There are two types of guaranteed delivery events:

- **GD Start events** occur when a client uses guaranteed delivery to invoke a service on a remote SAP BC Server. In a flow service, executing the pub.remote.gd:start service generates a GD Start event.

- **GD End events** occur when a client receives the results of the service it requested using guaranteed delivery. In a flow service, executing the pub.remote.gd:end service generates a GD End event.

Each guaranteed delivery transaction generates a GD Start event and a GD End event.

You can subscribe to GD Start and GD End events to trigger event handlers that log guaranteed delivery transactions to a file or database. You might also want to use guaranteed delivery events to trigger event handlers that send notification. For example, if you use guaranteed delivery to invoke a service that processes purchase orders, you might want to send notification to a business account manager about purchase orders from a particular client, or when the value of a purchase order is greater than a certain amount.

Guaranteed Delivery Events and Transaction Events

Guaranteed delivery events are related to transaction events (Tx Start and Tx End). Guaranteed delivery events begin when a client requests a guaranteed delivery transaction (GD Start) and when the client receives the results of the guaranteed delivery transaction (GD End). Transaction events occur when a service invoked using guaranteed delivery begins executing (Tx Start event) and when the service finishes executing (Tx End event).

The following diagram illustrates when guaranteed delivery events and transaction events occur during a guaranteed delivery transaction. In the following scenario, a local SAP BC Server uses guaranteed delivery to invoke a service on a remote server.
A Guaranteed Delivery Transaction generates Guaranteed Delivery Events and Transaction Events

1. Service A uses guaranteed delivery to invoke Service B on the remote SAP BC Server. When the local server requests Service B, the local server generates a GD Start event. By default, the GD Start event is logged to the txout.log file.

2. The remote SAP BC Server receives the request and begins executing Service B. When the remote server begins executing Service B, the remote server generates a Tx Start event. By default, the Tx Start event is logged to the txin.log file.

3. The remote SAP BC Server finishes executing Service B and generates a Tx End event. By default, the Tx End event is logged to the txin.log file.

4. The remote SAP BC Server sends the results of Service B to the requesting client (here, the local SAP BC Server).

5. The local SAP BC Server receives the results of Service B and generates a GD End event. By default, the GD End event is logged to the txout.log file.

For more information about guaranteed delivery, see “Using Guaranteed Delivery” on page 449.

Building Handlers for Guaranteed Delivery Start Events

When the Event Manager invokes an event handler for a GD Start event, the event handler receives an IData object containing the following run-time data. (A Specification and a record for these parameters reside in the pub.event folder.)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>A String containing the date and time that the event occurred, in the format yyyy/MM/dd HH:mm:ss SS</td>
</tr>
<tr>
<td>TID</td>
<td>A String containing the transaction identification number of the service that generated the GD Start event.</td>
</tr>
</tbody>
</table>
CHAPTER 21 Subscribing to Events

Building Handlers for Guaranteed Delivery End Events

When the Event Manager invokes an event handler for an GD End event, the event handler receives an IData object containing the following run-time data. (A Specification and a record for these parameters reside in the pub.event folder.)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>A String containing the date and time that the event occurred, in the format yyyy/MM/dd HH:mm:ss.SS</td>
</tr>
<tr>
<td>TID</td>
<td>A String containing the transaction identification number of the service that generated the GD End event.</td>
</tr>
<tr>
<td>result</td>
<td>A String containing the status of the guaranteed delivery transaction, such as DONE.</td>
</tr>
</tbody>
</table>

Tip! When you subscribe an event handler to a GD Start event, you can create a filter for the svcName field to specify the services in a guaranteed delivery transaction that you want to subscribe to—that is, you can specify the services that when invoked using guaranteed delivery will trigger the event handler.

Default GD Start and GD End Event Handler

By default, SAP BC Server is shipped with one GD Start and GD End event handler: pub.event.gd:logToFile. This event handler subscribes to all GD Start and GD End events. It is the mechanism that the server uses to log guaranteed delivery information to the <sapbc>\server\logs\txout.log file. In the txout.log, operations that begin with CREATE correspond to GD Start events. Operations that begin with END correspond to GD End events. Unless you explicitly want to eliminate or replace this server behavior, do not delete or modify this subscription. For more information about the txout.log file, see the SAP BC Administration Guide.
Working with Port Status Events

A port status event occurs each time SAP BC Server updates the server statistics. The port status event provides current status information about all of the configured ports on SAP BC Server.

You can use port status events to trigger services that send port status data to a network monitoring system. You can also use port status events to trigger services that write port status data to a log file.

Note: The `watt.server.stats.pollTime` property determines the frequency with which SAP BC Server updates server statistics. The default frequency is 10 seconds. For more information about this property, see the SAP BC Administration Guide.

Building Handlers for Port Status Events

When the Event Manager invokes an event handler for a port status event, the event handler receives an IData object that contains the following run-time data. (A specification and a record with these parameters are provided in the pub.event folder.)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>portStatusInfo</code></td>
<td>A Record List containing status information for each configured port on SAP BC Server.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>time</code></td>
<td>A String containing the date and time that the event occurred, in the format <code>yyyy/MM/dd HH:mm:ss.SS</code></td>
</tr>
<tr>
<td><code>port</code></td>
<td>A String containing the number for the port.</td>
</tr>
<tr>
<td><code>status</code></td>
<td>A String indicating the status of the port.</td>
</tr>
<tr>
<td><code>protocol</code></td>
<td>A String indicating the type of port (for example, http, https, ftp, or e-mail).</td>
</tr>
<tr>
<td><code>primary</code></td>
<td>A String indicating the primary port. By default, SAP BC Server designates an HTTP port at port 5555 as the primary port.</td>
</tr>
<tr>
<td><code>enabled</code></td>
<td>A String indicating whether or not the port is enabled. The value will be one of the following:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>true</code></td>
<td>The port is enabled</td>
</tr>
<tr>
<td><code>false</code></td>
<td>The port is disabled</td>
</tr>
</tbody>
</table>
Working with Replication Events

A replication event occurs when the `pub.replicator:generateReplicationEvent` service executes. You might want to generate and subscribe to replication events to trigger event handlers that automate the completion of the package replication and distribution processes. For example, you could create replication event handlers that do the following:

- Notify package subscribers when a package is published
- Maintain a log of replicated packages
- Maintain a log of the packages distributed or “pushed” to your subscribers
- Maintain a log of the packages your partners pulled from you

For more information about the `pub.replicator:generateReplicationEvent` service, see the SAP BC Built-In Services Guide.

Building Handlers for Replication Events

When the Event Manager invokes an event handler for a replication event, the event handler receives an `IData` object that contains the following run-time data. (A specification and a record for these parameters are provided in the `pub.event` folder.)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>A String containing the date and time that the event occurred, in the format <code>yyyy/MM/dd HH:mm:ss.SS</code></td>
</tr>
<tr>
<td>action</td>
<td>A user-defined String describing the action (such as create or push) for the replication event. You can use the value of the <code>action</code> variable to maintain separate logs for each action type.</td>
</tr>
<tr>
<td>package</td>
<td>A String containing the name of the released or pushed package.</td>
</tr>
<tr>
<td>service</td>
<td>A String containing the name of the flow service that invoked the <code>pub.replicator:generateReplicationEvent</code> service.</td>
</tr>
</tbody>
</table>

Tip! When you subscribe an event handler to a replication event, you can create a filter to specify the package that when replicated, will trigger the event handler.

Working with Session Events

A session event occurs when a client starts or ends a session on SAP BC Server or when SAP BC Server terminates an inactive session. You can subscribe to any of the following types of session events:
Session Start events occur when a developer uses Developer to open a session on SAP BC Server or when an <sapbc> client opens a session on the server to execute services.

Session End events occur when a developer or <sapbc> client specifically issues a disconnect instruction to SAP BC Server.

Session Expire events occur when SAP BC Server terminates an inactive session.

You can subscribe to session events to trigger event handlers that maintain your own log files or to trigger event handlers that send notification about users opening sessions on the server.

Building Handlers for Session Start Events

When the Event Manager invokes an event handler for a session start event, the event handler receives an IData object containing the following run-time data. (A Specification and a record for these parameters reside in the pub.event folder.)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>A String containing the date and time the event occurred, in the format yyyy/MM/dd HH:mm:ss.SS</td>
</tr>
<tr>
<td>sessionID</td>
<td>A String containing the identification number of the session.</td>
</tr>
<tr>
<td>userid</td>
<td>A String containing the user ID that the &lt;sapbc&gt; client or developer used to log on to SAP BC Server.</td>
</tr>
<tr>
<td>sessionName</td>
<td>A String containing the name of the new session.</td>
</tr>
</tbody>
</table>

Tip! When you subscribe an event handler to a Session Start event, you can create a filter so that only session start events generated by a specific user or by a member of a specific group trigger the event handler.

Default Session Start Event Handler

By default, SAP BC Server is shipped with one session start event handler: pub.event.sessionStartLogToFile. This event handler subscribes to all session start events. It is the mechanism that the server uses to log session information (including session end and session expire) to the <sapbc>\server\logs\session.log file. Unless you explicitly want to eliminate or replace this server behavior, do not delete or modify this subscription.

For more information about the session.log file, see the SAP BC Administration Guide.
Building Handlers for Session End Events

When the Event Manager invokes an event handler for a session end event, the event handler receives an IData object containing the following run-time data. (A Specification and a record for these parameters reside in the pub.event folder.)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>A String containing the date and time that the event occurred, in the</td>
</tr>
<tr>
<td></td>
<td>format yyyy/MM/dd HH:mm:ss.SS</td>
</tr>
<tr>
<td>sessionID</td>
<td>A String containing the identification number of the session.</td>
</tr>
<tr>
<td>rpcs</td>
<td>A String containing the number of service calls performed during the</td>
</tr>
<tr>
<td></td>
<td>session.</td>
</tr>
<tr>
<td>age</td>
<td>A String identifying how long the session existed (in milliseconds)</td>
</tr>
<tr>
<td></td>
<td>before it ended.</td>
</tr>
</tbody>
</table>

The Default Session End Event Handler

By default, SAP BC Server is shipped with one session end event handler: pub.event.sessionEnd.logToFile. This event handler subscribes to all session end events. It is the mechanism that the server uses to log session information (including session start and session expire) into the `<sapbc>\server\logs\session.log` file. Unless you explicitly want to eliminate or replace this server behavior, do not delete or modify this subscription.

For more information about the session.log file, see the SAP BC Administration Guide.

Building Handlers for Session Expire Events

When the Event Manager invokes an event handler for a session expire event, the event handler receives an IData object containing the following run-time data. (A Specification and a record for these parameters reside in the pub.event folder.)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>A String containing the date and time the event occurred, in the format</td>
</tr>
<tr>
<td></td>
<td>yyyy/MM/dd HH:mm:ss.SS</td>
</tr>
<tr>
<td>sessionID</td>
<td>A String containing the identification number of the session.</td>
</tr>
<tr>
<td>rpcs</td>
<td>A String containing the number of service calls performed during the</td>
</tr>
<tr>
<td></td>
<td>session.</td>
</tr>
<tr>
<td>age</td>
<td>A String identifying how long the session existed (in milliseconds)</td>
</tr>
<tr>
<td></td>
<td>before it expired.</td>
</tr>
</tbody>
</table>
Default Session Expire Event Handler

By default, SAP BC Server is shipped with one session expire event handler: pub.event.sessionExpire:logToFile. This event handler subscribes to all session expire events. It is the mechanism that the server uses to log session information (including session start and session end) to the <sapbc>\server\logs\session.log file. Unless you explicitly want to eliminate or replace this server behavior, do not delete or modify this subscription.

For more information about the session.log file, see the SAP BC Administration Guide.

Working with Stat Events

A stat event occurs each time SAP BC Server updates the statistics log (stats.log). The statistics log maintains statistical information about the consumption of system resources. The watt.server.stats.pollTime property determines the frequency with which SAP BC Server updates statistics. The default frequency is 10 seconds.

You can use stat events to trigger event handlers that maintain your own log file or to trigger event handlers that send server statistics to a network monitoring system.

Note: SAP BC Server provides an agent that you can configure for use with a network monitoring system. For information about implementing this agent, see the readme file in the agentInstall.jar file located in the <sapbc>\server\lib directory.

Building Handlers for Stat Events

When the Event Manager invokes an event handler for a stat event, the event handler receives an IData object containing the following run-time data. (A Specification and a record for these parameters reside in the pub.event folder.)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startTime</td>
<td>A String containing the date and time that the event occurred, in the format yyyy/MM/dd HH:mm:ss.SS</td>
</tr>
<tr>
<td>uptime</td>
<td>A String identifying the length of time the server has been running, in the format yyyy/MM/dd HH:mm:ss.SS</td>
</tr>
<tr>
<td>totalMem</td>
<td>A String identifying the total amount of used and unused storage space available (in kilobytes) to SAP BC Server.</td>
</tr>
<tr>
<td>freeMem</td>
<td>A String identifying the amount of unused storage space available (in kilobytes) to SAP BC Server.</td>
</tr>
<tr>
<td>usedMem</td>
<td>A String identifying the amount of storage used (in kilobytes) by SAP BC Server.</td>
</tr>
<tr>
<td>freeMemPer</td>
<td>A String identifying the percentage of free memory.</td>
</tr>
</tbody>
</table>
CHAPTER 21  Subscribing to Events

Default Stat Event Handler

By default, SAP BC Server is shipped with one stat event handler: pub.event.stats:logToFile. This event handler subscribes to all stat events. It is the mechanism that the server uses to log information to the \<sapbc>\server\logs\stats.log file. Unless you explicitly want to eliminate or replace this server behavior, do not delete or modify this subscription.

Working with Transaction Events

A transaction event occurs when a SAP BC Server begins and finishes executing a guaranteed delivery transaction. There are two types of transaction events:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>usedMemPer</td>
<td>A String identifying the percentage of used memory.</td>
</tr>
<tr>
<td>svrT</td>
<td>A String identifying the number of executing services.</td>
</tr>
<tr>
<td>svrTMax</td>
<td>A String identifying the maximum number of services that executed concurrently during the previous poll cycle.</td>
</tr>
<tr>
<td>sysT</td>
<td>A String identifying the number of threads in use.</td>
</tr>
<tr>
<td>sysTMax</td>
<td>A String identifying the maximum number of threads that executed concurrently during the previous poll cycle.</td>
</tr>
<tr>
<td>conn</td>
<td>A String identifying the number of current sessions on SAP BC Server.</td>
</tr>
<tr>
<td>connMax</td>
<td>A String identifying the maximum number of connections that ran concurrently during the previous poll cycle.</td>
</tr>
<tr>
<td>reqTotal</td>
<td>A String identifying the total number of requests during the poll cycle.</td>
</tr>
<tr>
<td>reqAvg</td>
<td>A String identifying the average processing duration for a service during the previous poll cycle.</td>
</tr>
<tr>
<td>newReqPM</td>
<td>A String identifying the new requests per minute at the beginning of the poll cycle.</td>
</tr>
<tr>
<td>endReqPM</td>
<td>A String identifying the new requests per minute at the end of the poll cycle.</td>
</tr>
<tr>
<td>errSvc</td>
<td>A String identifying the number of services that completed with errors since SAP BC Server started.</td>
</tr>
<tr>
<td>svcRate</td>
<td>A String identifying the number of service starts and ends per second during the last poll cycle.</td>
</tr>
<tr>
<td>errSys</td>
<td>A String identifying the number of errors that were not caused by services in the previous poll cycle.</td>
</tr>
</tbody>
</table>
- **Tx Start events** occur when a SAP BC Server begins executing a service invoked with guaranteed delivery.

- **Tx End events** occur when a SAP BC Server finishes executing a service invoked with guaranteed delivery.

Transaction events result from guaranteed delivery transactions. Each guaranteed delivery transaction generates a Tx Start event and a Tx End event. In fact, the transaction events occur between the guaranteed delivery events. A Tx Start event occurs immediately after a GD Start event and a Tx End event occurs immediately before a GD End event. For more information about how transaction events relate to guaranteed delivery events, see “Guaranteed Delivery Events and Transaction Events” on page 542.

You can subscribe to Tx Start and Tx End events to trigger event handlers that log guaranteed delivery transactions to a file or database. You might also want to use transaction events to trigger event handlers that send notification.

### Building Handlers for Transaction Start Events

When the Event Manager invokes an event handler for a Tx Start event, the event handler receives an IData object containing the following run-time data. (A Specification and a record for these parameters reside in the pub.event folder.)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>A String containing the date and time that the event occurred, in the format <code>yyyy/MM/dd HH:mm:ss,SS</code></td>
</tr>
<tr>
<td>TID</td>
<td>A String containing the transaction ID for the guaranteed delivery transaction that generated the event.</td>
</tr>
<tr>
<td>result</td>
<td>A String containing the status of the guaranteed delivery transaction, such as <code>NEW</code>.</td>
</tr>
</tbody>
</table>

### Building Handlers for Transaction End Events

When the Event Manager invokes an event handler for a Tx End event, the event handler receives an IData object containing the following run-time data. (A specification and a record for these parameters reside in the pub.event folder.)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>A String containing the date and time that the event occurred, in the format <code>yyyy/MM/dd HH:mm:ss,SS</code>.</td>
</tr>
</tbody>
</table>
Default Tx Start and Tx End Event Handler

By default, SAP BC Server is shipped with one Tx Start and Tx End event handler: pub.event.tx:logToFile. This event handler subscribes to all Tx Start and Tx End events. It is the mechanism that the server uses to log guaranteed delivery transaction information to the <sapbc>/server/logs/txin.log file. In the txin.log, operations that begin with CREATE correspond to Tx Start events. Operations that begin with END correspond to Tx End events. Unless you explicitly want to eliminate or replace this server behavior, do not delete or modify this subscription.

For more information about the txin.log, see the SAP BC Administration Guide.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TID</td>
<td>A String containing the transaction ID of the guaranteed delivery transaction that generated the event.</td>
</tr>
<tr>
<td>result</td>
<td>A String containing the status of the guaranteed delivery transaction, such as DONE.</td>
</tr>
</tbody>
</table>
SAP BC Flow Steps

- BRANCH ................................................................. 554
- EXIT ................................................................. 557
- INVOKE ............................................................... 559
- LOOP ................................................................. 561
- MAP ................................................................. 563
- REPEAT .............................................................. 565
- SEQUENCE ......................................................... 568
The BRANCH step selects and executes a child step based on the value of one or more variables in the pipeline. You specify the variables you want to branch on by specifying a switch value or by writing an expression that includes the variables.

**Branching on a Switch Value**

When you branch on a switch value, you specify the switch variable in the `switch` property of the BRANCH step. In the `label` property for each child step, you specify the value of the switch variable that will trigger that child step. At run time, the BRANCH flow step executes the child step that has the same label as the value of the `switch` property.

If you want to execute a child step when the value of the `switch` property is an empty string, leave the `label` property of the child step blank. If you want to execute a child step when the `switch` property is a null or unmatched string, set the `label` of the child step to `$null` or `$default`.

**Branching on Expressions**

When you branch on expressions, you set the `evaluate-labels` property of the BRANCH step to true. In the `label` property for each child step, you write an expression that includes one or more variables. At run time, the BRANCH step executes the first child step with an expression that evaluates to true.

If you want to specify a child step to execute when none of the expressions are true, set the label of the child step to `$default`.

```
if the value of choice is 'Name1'
  Name1
if the value of choice is 'Name2'
  Name2
  *
  *
if the value of choice is 'NameN'
  NameN
Otherwise
  *
```

Name of the switch field is choice
The BRANCH step in the following illustration uses expressions to determine which of the four children executes. The value of the subTotal variable determines which step executes at run time.

Properties

The BRANCH step has the following properties.

comment
Optional. Specifies a descriptive comment for the step.

scope
Optional. Specifies the name of a record in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this field blank.
timeout
Optional. Specifies the maximum number of seconds that the step should run. When the time is exceeded, the step stops and raises an exception. However, if this step contains an INVOKE step, the INVOKE step is not interrupted when the timeout value is exceeded. Instead, the exception is raised after the INVOKE is complete.

If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %expiration%.

If you do not want to specify a timeout period, set timeout to zero or leave it blank.

dlabeled
Optional. (Required if you are using this BRANCH step as a target for another BRANCH or EXIT step.) Specifies a name for this instance of the BRANCH step, or a null, unmatched, or empty string ($null, $default, blank).

switch
Specifies the name of the string variable in the pipeline that the BRANCH step uses as the switch field to determine which child step is executed at run time. Do not specify a variable for switch if you plan to branch on expressions.

evaluate-labels
Specifies whether or not you want to branch on expressions. When you branch on expressions, you enter expressions in the label field for the children of the BRANCH step. At run time, the server executes the first child step whose label evaluates to true. Select true to branch on expressions. To branch on the switch value, select false.

Conditions That Will Cause a BRANCH Step to Fail

- The switch field is not in the pipeline.
- The matching child step fails.
- The BRANCH step does not complete before the timeout period expires.
EXIT

The EXIT step exits the entire flow service or a single flow step. Specifically, it may exit from the nearest ancestor loop step, a specified ancestor step, the parent step, or the entire flow service.

The EXIT step can throw an exception if the exit is considered a failure. When an exception is thrown, user-specified error message text is displayed by typing it directly or by assigning it to a variable in the pipeline.

Properties

The EXIT step has the following properties.

- comment: Optional. Specifies a descriptive comment for the step.
- label: Optional. (Required if you are using this EXIT step as a target for a BRANCH step.) Specifies a name for this specific step, or a null, unmatched, or empty string ($null, $default, blank).
- from: Required. Specifies the step that you want to exit from.

Specify this value... To exit the...

- $loop: Nearest ancestor LOOP or REPEAT step.
- $parent: Parent flow step, regardless of the type of step.
- $flow: Entire flow.
- label: Nearest ancestor step that has a label that matches this value.

Note: If the label you specify does not match the label of an ancestor flow step, the flow will exit with an exception.

- signal: Required. Specifies whether the exit is to be considered a success or a failure. A SUCCESS condition exits the flow service or step. A FAILURE condition exits the flow service or step and throws an exception. The text of the exception message is contained in the failure-message property.

- failure-message: Optional. Specifies the text of the exception message that is displayed when signal is set to FAILURE. If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %mymessage%.
Examples of When to Use

- Exit an entire flow service from within a series of deeply nested steps.
- Throw an exception when you exit a flow service or a flow step without having to write a Java service to call Service.throwError() .
- Exit a LOOP or REPEAT flow step without throwing an exception.
The INVOKE flow step invokes another service. You can use it to invoke any type of service, including another flow service.

Properties

The INVOKE step has the following properties.

comment
Optional. Specifies a descriptive comment for the step.

scope
Optional. Specifies the name of a record in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this field blank.

timeout
Optional. Specifies the maximum number of seconds that the step should run. When the time is exceeded, the step stops and raises an exception. However, the INVOKE step is not interrupted when the timeout value is exceeded. Instead, the exception is raised after the INVOKE is complete.

If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %expiration%.

If you do not want to specify a timeout period, set timeout to zero or leave it blank.

label
Optional. (Required if you are using this step as a target for a BRANCH or EXIT step.) Specifies a name for this specific step, or a null, unmatched, or empty string ($null, $default, blank).

service
Required. Specifies the fully qualified name of the service to invoke.

validate-in
Optional. Specifies validation of the input to the invoked service. If you want the input to be validated against the input parameters of the service, select $default.

validate-out
Optional. Specifies validation of the output from the invoked service. If you want the output to be validated against the output parameters of the service, select $default.

Note: In SAP BC version 3.x, you could set an as-user property for a transformer. In SAP BC Server version 4.0, this property was removed. For more information see “The As-User Property in SAP BC Developer 3.x” on page 97.
Conditions That Will Cause an INVOKE Step to Fail

- The service that is invoked fails.
- The specified service does not exist.
- The specified service is disabled.
The LOOP step takes as input an array variable that is in the pipeline. It loops over the members of an input array, executing its child steps each time through the loop. For example, you have a service that takes a string as input and a string list in the pipeline. Use the LOOP step to invoke the service one time for each string in the string list.

You identify a single array variable to use as input when you set the properties for the LOOP step. You can also designate a single variable for output. The LOOP step collects an output value each time it runs through the loop and creates an output array that contains the collected output values. If you want to collect more than one variable, specify a record that contains the fields you want to collect for the output variable.

The LOOP Step

Properties

The LOOP step has the following properties.

comment
Optional. Specifies a descriptive comment for the step.

scope
Optional. Specifies the name of a record in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this field blank.

timeout
Optional. Specifies the maximum number of seconds that the step should run. When the time is exceeded, the step stops and raises an exception. However, if this step contains an INVOKE step, the INVOKE step is not interrupted when the timeout value is exceeded. Instead, the exception is raised after the INVOKE is complete.

If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %expiration%.
If you do not want to specify a timeout period, set `timeout` to zero or leave it blank.

**label**
Optional. (Required if you are using this step as a target for a BRANCH or EXIT step.)
Specifies a name for this specific step, or a null, unmatched, or empty string ($null, $default, blank).

**in-array**
Required. Specifies the input array over which to loop. You must specify a variable in the pipeline that has an array data-type; that is, string list, string table, record list, or object list.

**out-array**
Optional. Specifies the name of the output variable. The value of this variable is collected each time through the loop and placed in an output array of this name. You do not need to specify this property if the loop does not produce output values.

**Conditions That Will Cause a LOOP Step to Fail**

- The pipeline does not contain the input array.
- The input variable is not an array variable.
- A child step of the LOOP step fails during any iteration of the loop.
- The LOOP step does not complete before the timeout period expires.
The MAP step adjusts the pipeline at any point in a flow. It makes pipeline modifications that are independent of an INVOKE step.

Within the MAP step, you can:
- Map (copy) the value of a pipeline input variable to a new or existing pipeline output variable.
- Drop an existing pipeline input variable. (Keep in mind that once you drop a variable from the pipeline, it is no longer available to subsequent services in the flow.)
- Assign a value to a pipeline output variable.
- Perform document-to-document mapping in a single view by inserting transformers.

**Properties**

The MAP step has the following properties.

- **comment**
  - Optional. Specifies a descriptive comment for this step.

- **scope**
  - Optional. Specifies the name of a record in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this field blank.

- **timeout**
  - Optional. Specifies the maximum number of seconds that this step should run. When the time is exceeded, the step stops and raises an exception. However, if this step contains an INVOKE step (a transformer is essentially an INVOKE step), the INVOKE step is not interrupted when the timeout value is exceeded. Instead, the exception is raised after the INVOKE is complete.
  - If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %expiration%.
  - If you do not want to specify a timeout period, set **timeout** to zero or leave it blank.

- **label**
  - Optional. (Required if you are using this step as a target for a BRANCH or EXIT step.) Specifies a name for this specific step, or a null, unmatched, or empty string ($null, $default, blank).
Example of When to Use

- You want to assign an initial set of input values in a flow service (i.e., initialize - variables). You insert the MAP step at the beginning of the flow, and then use the Set Value modifier to assign values to the appropriate variables in Pipeline Out.

- You want to map a document from one format to another (for example, cXML to XML). Insert transformers into the MAP step to perform the needed data transformations. For more information about transformers, see “What Are Transformers?” on page 177.
The REPEAT step repeatedly executes its child steps up to a maximum number of times that you specify. It determines whether to re-execute the child steps based on a repeat on condition. You can set the repeat condition to one of the following:

- Repeat if any one of the child steps fails
- Repeat if all of the elements succeed

You can also specify a back-off time period that you want the REPEAT flow step to wait before it re-executes its child steps.

**Properties**

The REPEAT step has the following properties.

**comment**
Optional. Specifies a descriptive comment for this step.

**scope**
Optional. Specifies the name of a record in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this field blank.

**timeout**
Optional. Specifies the maximum number of seconds that this step should run. When the time is exceeded, the step stops and raises an exception. However, if this step contains an INVOKE step, the INVOKE step is not interrupted when the timeout value is exceeded. Instead, the exception is raised after the INVOKE is complete.
If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %expiration%.

If you do not want to specify a timeout period, set timeout to zero or leave it blank.

**label**
Optional. (Required if you are using this step as a target for a BRANCH or EXIT step.) Specifies a name for this specific step, or a null, unmatched, or empty string ($null, $default, blank).

**count**
Required. Specifies the number of times the REPEAT step re-executes its child steps while repeat-on is true. Note that count specifies the maximum number of times that the set of child steps re-execute. They are always executed once. For example, if you set count to 0, children are executed once but cannot be re-executed. If you set count to 1, children are executed once, and can be re-executed once more (max) if the repeat-on condition is true.

If you want the children re-executed as long as the specified repeat-on condition remains true (i.e., no maximum limit), set this value to -1.

If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %servicecount%.

**backoff**
Optional. Specifies the number of seconds to wait between attempts to execute the child steps. Specify zero (0) to re-execute the child steps without a delay.

If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %waittime%.

**repeat-on**
Required. Specifies the condition that causes the REPEAT step to re-execute its child steps. Specify one of the following:

- **FAILURE** - The REPEAT step re-executes the child steps if any of the child steps fail. This is the default.
- **SUCCESS** - The REPEAT step re-executes the child steps if all of the child steps succeed.

### When Does REPEAT Fail?

The following conditions cause the REPEAT step to fail:

<table>
<thead>
<tr>
<th>repeat on condition</th>
<th>The step fails if...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SUCCESS</strong></td>
<td>A child within the REPEAT block fails.</td>
</tr>
<tr>
<td><strong>FAILURE</strong></td>
<td>The count limit is reached before its children execute successfully.</td>
</tr>
</tbody>
</table>

If the REPEAT step is a child of another step, the failure is propagated to its parent.
Examples of When to Use

- **Repeat-on condition set to FAILURE**—Use when a service accesses a remote server and you want the service to retry if the server is busy. Make the service that accesses the remote server a child element of a REPEAT flow step, and then set the repeat-on condition to FAILURE. If the service attempts to access the Web site and it fails, the REPEAT flow step attempts to retry the service again. You also set a back-off time that causes the REPEAT flow condition to wait a period of time before invoking the service again.

- **Repeat-on condition set to SUCCESS**—Use in a Web-automation service, when you want to repeat a load and query step as long as a “Next Page” button exists in the current document, indicating that there are additional pages to be processed. You implement a REPEAT flow step, ending the loop when the query step “fails” to retrieve a “Next Page” button in the current document.
SEQUENCE

The SEQUENCE step forms a collection of child steps that execute sequentially. This is useful when you want to group a set of steps as a target for a BRANCH step.

You can set an exit condition that indicates whether the sequence should exit prematurely, and if so, under what condition. Specify one of the following exit conditions:

- **Exit the sequence when a child step fails.** Use this condition when you want to ensure that all child steps are completed successfully. If any child step fails, the sequence ends prematurely and the sequence fails.

- **Exit the sequence when a child step succeeds.** Use this condition when you want to define a set of alternative services, so that if one fails, another is attempted. If a child step succeeds, the sequence ends prematurely and the sequence succeeds.

- **Exit the sequence after executing all child steps.** Use this condition when you want to execute all of the child steps regardless of their outcome. The sequence does not end prematurely.

### The SEQUENCE Step

First... If exit condition is not met... If exit condition is not met...

First... If exit condition is not met... If exit condition is not met...

### Properties

The SEQUENCE step has the following properties.

- **comment**
  Optional. Specifies a descriptive comment for this step.

- **scope**
  Optional. Specifies the name of a record in the pipeline to which you want to restrict this step. If you want this step to have access to the entire pipeline, leave this field blank.

- **timeout**
  Optional. Specifies the maximum number of seconds that this step should run. When the time is exceeded, the step stops and raises an exception. However, if this step contains an INVOKE step, the INVOKE step is not interrupted when the timeout value is exceeded. Instead, the exception is raised after the INVOKE is complete.

  If you want to use the value of a pipeline variable for this property, type the variable name between % symbols. For example, %expiration%.

  If you do not want to specify a timeout period, set **timeout** to zero or leave it blank.
label
Optional. (Required if you are using this step as a target for a BRANCH or EXIT step.)
Specifies a name for this specific step, or a null, unmatched, or empty string ($null, $default, blank).

exit-on
Required. Indicates whether the SEQUENCE step should exit the sequence prematurely, and if so, under what condition it should exit. Specify one of the following:

**FAILURE**
Exit the sequence when a child step fails.
The SEQUENCE step executes its child steps until either one fails or until it executes all its child steps. This is the default.

**SUCCESS**
Exit the sequence when a child step succeeds.
The SEQUENCE step executes its child steps until either one succeeds or until it executes all its child steps.

**DONE**
Exit the sequence after executing all child steps.
The SEQUENCE step executes all of its child steps regardless of whether they succeed or fail.

**Conditions That Will Cause the SEQUENCE Step to Fail**
This section describes the conditions that cause failure based on the exit condition for the sequence.
If `exit-on` is set to FAILURE, conditions that will cause a failure include:
- One of the child steps fails.
- The SEQUENCE step does not complete before the timeout period expires.
If `exit-on` is set to SUCCESS, conditions that will cause a failure include:
- All the child steps fail.
- The SEQUENCE step does not complete before the timeout period expires.
If `exit-on` is set to DONE, conditions that will cause a failure include:
- The SEQUENCE step does not complete before the timeout period expires.
webMethods Query Language

- Overview ................................................................. 572
- Object References .................................................... 572
- Object Properties ..................................................... 575
- Property Masking ..................................................... 575
Overview

The webMethods Query Language (WQL) provides the primary mechanism for mapping data from Web documents. When a Web document is read by webMethods, the XML or HTML markup within the document is used to parse the contents of the document into the object model.

XML and HTML markup both consist of tag elements enclosed in angle brackets: < >. In the process of parsing, tag elements are transformed into arrays of objects; the attributes of tag elements become object properties. XML and HTML markup both implement containing elements and empty elements. Containing elements have open and close tags. Empty elements are single tags.

When a Web document is parsed, the text contained within containing elements becomes the text property of the corresponding document object.

Data parsed from Web documents is accessed with WQL queries, which consist of one or more indexed element arrays and an object property.

Object References

For the following object references, x and y represent numerical indexes.

- `doc.element[x].property`  
  An absolute reference uses a numerical index into an element array.

- `doc.element[x].element[x].property`  
  Nested element arrays scope the object reference to children elements.

- `doc.element[x].line[x].property`  
  An array of lines is fabricated when the text property of a document object contains line breaks.

- `doc.element[x].^property`  
  The parent of an element is referenced with ^.

- `doc.element[x].?x.property`  
  A ? matches any type of element array.

- `doc.element[].property`  
  Empty brackets signify that all members of an element array are to be returned.

- `doc.element[element].property`  
  The | is used to signify a logical 'OR'. The contents of two or more element arrays can be returned.

- `doc.element[x-y].property`  
  Returns a range of elements from an array.

- `doc.element[x-end].property`  
  end is a reserved word that returns the final element of an array.
**doc.element[x,y,z].property**
Returns items x, y, and z where x, y, and z represent numerical indexes into an element array.

**doc.element[x+y].property**
Returns element x and every y element thereafter.

**doc.element['match'].property**
Returns an array of elements whose text property matches the match string, which can contain the following wildcard characters:

**Use this Character...**    **To...**

* Match any sequence of zero or more characters

? Match any single character.

% Matches a single word, where word is defined to be any sequence of non-whitespace characters

\ Escape any of the above wildcard characters.

Match strings are compared with the .text property of the indexed object. The .text property contains the text of all child objects.

**doc.element[[RegularExpression]].property**
Returns an array of elements whose text property matches the specified regular expression. For information about how to construct a regular expression, see “Regular Expressions” on page 585.

**doc.element[property='match'].property**
Matches the value of a specific element property.
Sibling Operators

WQL provides the following set of operators to refer to siblings of a specified element. The examples shown in these descriptions refer to the following HTML structure:

```
  | TABLE
  |   TR
  |     TD
  |       B Bold 0
  |       I Italic 0
  |       B Bold 1
  |       B Bold 2
  |     TD
  |       B Bold 3
  |       I Italic 1
  |       B Bold 4
  |       I Italic 2

  Sibling elements belonging to the same parent,
```

The sibling operators are constrained by the current parent. If an operator exceeds the boundaries of the current parent, a null value is produced for that reference.

**doc.element[x].@n.property**

References the *n*th sibling after `element[x]`, regardless of type. Compare with `doc.element[x].+n.property`, below.

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>doc.td[0].b[0].@1.text</code></td>
<td>Italic 0</td>
</tr>
<tr>
<td><code>doc.td[0].i[0].@1.text</code></td>
<td>Bold 1</td>
</tr>
<tr>
<td><code>doc.td[0].b[0].@4.text</code></td>
<td>Null</td>
</tr>
</tbody>
</table>

**doc.element[x].@-n.property**

References the *n*th sibling prior to `element[x]`, regardless of type. Compare with `doc.element[x].-n.property`, below.

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>doc.td[1].b[end].@-2.text</code></td>
<td>Bold 3</td>
</tr>
<tr>
<td><code>doc.td[1].i[end].@-1.text</code></td>
<td>Bold 4</td>
</tr>
<tr>
<td><code>doc.td[1].b[end].@-3.text</code></td>
<td>Null</td>
</tr>
</tbody>
</table>
Object Properties

In addition to the properties derived from the attributes of a parsed XML or HTML tag element, the following properties are available for all objects:

.text/text
Returns the text of an object.

.value/val
Returns the value of an object. (Equivalent to the text of the object if the element has no VALUE attribute.)

.src/.src
Returns the XML or HTML source of an object.

.idx/.idx
Returns the numerical index of an object.

.ref/.ref
Returns a complete object reference.

Example Result

```
Example                  Result
doc.td[0].b[0].+1.text   Bold 1
doc.td[0].i[0].+1.text   Null
doc.td[0].b[0].+3.text   Null
```

Example Result

```
Example                  Result
doc.td[1].b[end].-2.text Null
doc.td[1].i[end].-1.text Italic 1
doc.td[1].b[end].-3.text Null
```
Property Masking

Property masking allows for the stripping away of unwanted text from the value of an object property.

`doc.element[x].property[x-y]`
Returns a range of characters from position x to y.

`doc.element[x].property['mask']`
Uses wildcard matching and token collecting to extract desired data from the value of an object property.

`doc.element[x].property[/RegularExpression/]`
Uses a regular expression to extract desired data from the value of an object property. For information about how to construct a regular expression, see “Regular Expressions” on page 585.
Supported Data Types

- Data Types ................................................................. 578
- Mapping Data Types ..................................................... 580
Data Types

Data is passed in and out of a service through an IData object. An IData object is the collection of name/value pairs on which a service operates. An IData object can contain any number of elements of any valid Java objects, including additional IData objects and IDataCodable objects.

Each element stored in an IData object corresponds to an SAP BC data type. The following table identifies the data types supported by Developer.

<table>
<thead>
<tr>
<th>SAP BC Data Type</th>
<th>Icon</th>
<th>Description</th>
<th>Java Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>![Icon]</td>
<td>String of characters.</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>String List</td>
<td>![Icon]</td>
<td>A one-dimensional String array.</td>
<td>java.lang.String[ ]</td>
</tr>
<tr>
<td>String Table</td>
<td>![Icon]</td>
<td>A two-dimensional String array.</td>
<td>java.lang.String[ ][ ]</td>
</tr>
<tr>
<td>Record</td>
<td>![Icon]</td>
<td>A data structure that is a container for other variables.</td>
<td>com.wm.data.IData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Records can contain variables of any other data type. The contents of a</td>
<td>com.wm.util.Values</td>
</tr>
<tr>
<td></td>
<td></td>
<td>record are stored as key/value pairs where the variable name is the key.</td>
<td></td>
</tr>
<tr>
<td>Record List</td>
<td>![Icon]</td>
<td>A one-dimensional array of Records (IData [ ]or Values [ ]).</td>
<td>com.wm.data.IData[ ]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>com.wm.util.Values[ ]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>com.wm.util.Table</td>
</tr>
<tr>
<td>Record Reference</td>
<td>![Icon]</td>
<td>A Record whose structure is defined by a Record in the Service Browser.</td>
<td>Reference to an existing object</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>which implements the com.wm.data.IData</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>interface or a reference to an</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>existing com.wm.util.Values object.</td>
</tr>
<tr>
<td>Record Reference List</td>
<td>![Icon]</td>
<td>A Record List whose structure is defined by a Record in the Service</td>
<td>Reference to an existing object</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Browser.</td>
<td>which implements the com.wm.data.IData</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>interface or a reference to an</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>existing com.wm.util.Values object.</td>
</tr>
</tbody>
</table>
### SAP BC Data Type Icon Description Java Data Type

<table>
<thead>
<tr>
<th>SAP BC Data Type</th>
<th>Icon</th>
<th>Description</th>
<th>Java Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>✴</td>
<td>Any data type that does not fall into any of the data types described in the above rows, will be shown as an Object.</td>
<td>Any subclass of java.lang.Object.</td>
</tr>
</tbody>
</table>

#### How SAP BC Server Handles Objects and Object Lists

SAP BC Server handles variables represented by Object or Object List icons “by reference”—that is, the variables represented by the ✴ and ✴✴ icons in the Pipeline Editor do not contain actual values. Instead, the variable stores an address or reference to the actual object. In contrast, SAP BC Server handles the other data types by value—the actual values are stored in the variables and passed to and from services.

**Note:** When mapping variables, SAP BC Server handles all data types except string by reference as well. For more information about mapping variables by reference, see “What Happens When SAP BC Server Executes a Map Between Variables at Run Time?” on page 160.

**Note:** For SAP BC Server built-in services, you can view the actual data types represented by Object or Object List icons by looking up the service in the SAP BC Built-In Services Guide.

#### How the SAP BC Developer Supports Tables

With the exception of String Table, Developer does not provide a separate data type for tables. However, tables can appear as Record Lists or Objects. Tables that are instances of com.wm.util.Table appear as Record Lists in SAP BC Developer. These tables can be used as Record Lists in flow services. Services in the WmDB package use tables that are instances of wm.com.util.Table.

Tables can also be declared as Objects. Objects or user-defined table-like objects that do not implement the com.wm.util.pluggable.WMIDataList interface appear as Objects in SAP BC Developer.
Values Objects and IData

In earlier versions of SAP BC Server, data was stored in a Values object. In SAP BC 3.5 and later, data is stored in an IData object. Current Values object (com.wm.util.Values class) is an implementation of IData and supports the IData features. Current com.wm.util.Values class also supports the old Values object behavior. For more information about IData and Values objects, see “The IData Object” on page 288, “The Values Object” on page 290, and the SAP BC Java API Reference in <sapbc>/developer/doc/api/index.html.

Mapping Data Types

In the Pipeline Editor, you can map variables of different, but compatible, data types to one another. The following table identifies which data types you can map to each other.

<table>
<thead>
<tr>
<th>You can map this data type...</th>
<th>To these data types....</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>T T T T * *</td>
</tr>
<tr>
<td>String List</td>
<td>T T T T * *</td>
</tr>
<tr>
<td>String Table</td>
<td>T T T T * *</td>
</tr>
<tr>
<td>Record</td>
<td></td>
</tr>
<tr>
<td>Record List</td>
<td></td>
</tr>
<tr>
<td>Record Reference</td>
<td></td>
</tr>
<tr>
<td>Record Reference List</td>
<td></td>
</tr>
<tr>
<td>Object</td>
<td>* T T T E E * *</td>
</tr>
<tr>
<td>Object List</td>
<td>E E E E * *</td>
</tr>
</tbody>
</table>

For detailed information and guidelines for mapping variables of different data types, see “Mapping Variables of Different Data Types” on page 163.

Default Pipeline Rules for Mapping to and from Array Variables

When you map between scalar and array variables, you can specify which element of the array variable you want to map to or from. Scalar variables are those that hold a single value, such as String, Record, and Object. Array variables are those that hold multiple
values, such as String List, String Table, Record List, and Object List. For example, you can map a String to the second element of a String List.

If you do not specify which element in the array variable that you want to map to or from, Developer uses the default rules in the Pipeline Editor to determine the value of the target variable. The following table identifies the default pipeline rules for mapping to and from array variables.

<table>
<thead>
<tr>
<th>If you map...</th>
<th>To...</th>
<th>Then...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A scalar variable</td>
<td>An array variable that is empty (the variable does not have a defined length)</td>
<td>The map defines the length of the array variable—it contains one element and has length of one. The first (and only) element in the array is assigned the value of the scalar variable.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>If you map...</th>
<th>To...</th>
<th>Then...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A scalar variable</td>
<td>An array variable with a defined length</td>
<td>The length of the array is preserved and each element of the array is assigned the value of the scalar variable.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>If you map...</th>
<th>To...</th>
<th>Then...</th>
</tr>
</thead>
<tbody>
<tr>
<td>An array variable</td>
<td>A scalar variable</td>
<td>The scalar variable is assigned the first element in the array.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>If you map...</th>
<th>To...</th>
<th>Then...</th>
</tr>
</thead>
</table>

If you map...  To...  Then...

value  [empty]  value

value  X  Y  Z  value  value  value  value

X  Y  Z  [empty]  X
An array variable  An array variable that does not have a defined length  The map defines the length of the target array variable—it will be the same length as the source array variable. The elements in the target array variable are assigned the values of the corresponding elements in the source array variable.

<table>
<thead>
<tr>
<th>If you map...</th>
<th>To...</th>
<th>Then...</th>
</tr>
</thead>
<tbody>
<tr>
<td>An array variable</td>
<td>An array variable that has a defined length</td>
<td>The length of the source array variable must equal the length of the target array variable. If the lengths do not match, the mapping will not occur. If the lengths are equal, the elements in the target array variable are assigned the values of the corresponding elements in the source array variable.</td>
</tr>
</tbody>
</table>

Note: A source variable that is the child of a Record List is treated like an array because there is one value of the source variable for each record in the Record List. For example:
### Mapping Data Types

#### If you map...
<table>
<thead>
<tr>
<th>RecordList1</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>String1</td>
<td>StringList1</td>
</tr>
</tbody>
</table>

#### Where the value of RecordList1 is... Then the value of StringList1 is...

- **RecordList1**
  - **RecordList1 [0]**
    - **String1** a
  - **RecordList1 [1]**
    - **String1** b
  - **RecordList1 [2]**
    - **String1** c
Regular Expressions

- What Is a Regular Expression? .................................................. 586
- Using a Regular Expression in a Mask ........................................ 586
- Regular Expression Operators .................................................. 587
What Is a Regular Expression?

A regular expression is a pattern-matching technique used extensively in UNIX environments. SAP BC Developer lets you use regular expressions to specify pattern-matching strings for some of its functions. For example, you can use a regular expression to specify an index, a property, or a mask in a webMethods Query Language (WQL) statement. You can also use a regular expression to specify the switch value for a BRANCH step or to describe a URL in a WebTap rule.

To specify a regular expression, you must enclose the expression between / symbols. When the server encounters this symbol, it knows to interpret the characters between these symbols as a pattern-matching string (i.e., a regular expression).

A simple pattern-matching string such as /string/ matches any element that contains string. So, for example, the regular expression /SAP BC/ would match all of the following strings:

“SAP BC”
“You use SAP BC Server to execute services”
“Exchanging data with XML is easy using SAP BC”
“SAP BC Server”

Important! Characters in regular expressions are case-sensitive.

Using a Regular Expression in a Mask

When you use a regular expression as a mask, you use parenthesis to specify which characters you want to collect. For example, the object reference:

   doc.p[].text[/(.{30}).*/]

retains the first 30 characters in each matching element and discards the rest.
## Regular Expression Operators

Following are the operators supported in SAP BC’s implementation of regular expressions.

<table>
<thead>
<tr>
<th>Use this symbol...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Match any single character except a new line.</td>
</tr>
<tr>
<td>^</td>
<td>Match the beginning of the string or line.</td>
</tr>
<tr>
<td>$</td>
<td>Match the end of the string or line.</td>
</tr>
<tr>
<td>*</td>
<td>Match the preceding item zero or more times.</td>
</tr>
<tr>
<td>+</td>
<td>Match the preceding item 1 or more times.</td>
</tr>
<tr>
<td>?</td>
<td>Match the preceding item 0 or 1 times.</td>
</tr>
</tbody>
</table>

**Example**  
`doc.p[/web.methods/].text`  
This example would return any paragraph containing the string ‘web’ followed by any single character and the string ‘methods’. It would match both ‘webMethods’ and ‘webmethods’.

**Example**  
`doc.p[/^webMethods/].text`  
This example would return any paragraph containing the string ‘webMethods’ at the beginning of the element or at the beginning of any line within that element.

**Example**  
`doc.p[/webMethods$/].text`  
This example would return any paragraph containing the string ‘webMethods’ at the end of the paragraph element or at the end of any line within that element.

**Example**  
`doc.p[/part *555-A/].text`  
This example would return any paragraph containing the string ‘part’ followed by zero or more spaces and then the characters ‘555-A’.

**Example**  
`doc.p[/part number +555-A/].text`  
This example would return any paragraph containing the string ‘part’ followed by one or more spaces and then the characters ‘555-A’.

**Example**  
`doc.p[/part ?555-A/].text`  
This example would return any paragraph containing the string ‘part number’ followed by zero or one space and then the characters ‘555-A’.
<table>
<thead>
<tr>
<th>Use this symbol...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>When used in an index, these characters group an item within the regular expression.</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong> <em>doc.p[/part(,0)May+]/.text</em></td>
</tr>
<tr>
<td></td>
<td>This example would return any paragraph containing the string ‘part’ followed by one or more occurrences of the characters ‘,0’ and then the characters ‘May’.</td>
</tr>
<tr>
<td></td>
<td>When used in a mask, they specify characters that you want to retain.</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong> <em>doc.p[].text[(^.{25}).</em>]*</td>
</tr>
<tr>
<td></td>
<td>This example would keep the first 25 characters within each paragraph and discard the rest.</td>
</tr>
<tr>
<td>{n}</td>
<td>Match the preceding item exactly <em>n</em> times.</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong> <em>doc.p[/^.{24}webmethods/].text</em></td>
</tr>
<tr>
<td></td>
<td>This example would return any paragraph in which the word ‘webmethods’ started in the 25th character position of the paragraph.</td>
</tr>
<tr>
<td>{n,}</td>
<td>Match the preceding item <em>n</em> or more times.</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong> <em>doc.p[/^.{10,}webmethods/].text</em></td>
</tr>
<tr>
<td></td>
<td>This example would return any paragraph in which the word ‘webmethods’ appeared anywhere after the 10th character position of the paragraph.</td>
</tr>
<tr>
<td>{0,m}</td>
<td>Match the preceding item none or at most <em>m</em> times.</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong> <em>doc.p[/^.{0,4}webmethods/].text</em></td>
</tr>
<tr>
<td></td>
<td>This example would return any paragraph in which the word ‘webmethods’ started in any of the first 5 character positions of the paragraph.</td>
</tr>
<tr>
<td>{n, m}</td>
<td>Match the preceding item at least <em>n</em> times, but not more than <em>m</em> times.</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong> <em>doc.p[/^.{1,4}webmethods/].text</em></td>
</tr>
<tr>
<td></td>
<td>This example would return any paragraph in which the word ‘webmethods’ started in character position 2 through 5 of the paragraph.</td>
</tr>
<tr>
<td></td>
<td>Match the expression that precedes or follows this character.</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong> *doc.p[/webmethods</td>
</tr>
</tbody>
</table>
| | This example would return any paragraph that contained either ‘webmethods’ or ‘webMethods’.
<table>
<thead>
<tr>
<th>Use this symbol...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>Match a word boundary.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>doc.p[/\bport\b/].text</td>
</tr>
<tr>
<td></td>
<td>This example would return any paragraph that contained the word ‘port’, but not paragraphs that contained these characters as part of a larger word, such as ‘import’, ‘support’, ‘ports’ or ‘ported’.</td>
</tr>
<tr>
<td>\B</td>
<td>Match a boundary that is not a word boundary.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>doc.p[/\B555-A/].text</td>
</tr>
<tr>
<td></td>
<td>This example would return any paragraph that contained the characters ‘555-A’ as part of a larger word such as AZ555-A, or Dept555-A, but not ‘555-A’ alone.</td>
</tr>
<tr>
<td>\A</td>
<td>Match only at the beginning of a string (equivalent to ^).</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>doc.p[/\AwebMethods/].text</td>
</tr>
<tr>
<td></td>
<td>This example would return any paragraph containing the string ‘webMethods’ at the beginning of the element or at the beginning of any line within that element.</td>
</tr>
<tr>
<td>\Z</td>
<td>Match only at the end of a string (or before a new line at the end).</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>doc.p[/webMethods\Z/].text</td>
</tr>
<tr>
<td></td>
<td>This example would return any paragraph containing the string ‘webMethods’ at the end of the paragraph element or at the end of any line within that element.</td>
</tr>
<tr>
<td>\n</td>
<td>Match a new line.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>doc.p[/webMethods\n/].text</td>
</tr>
<tr>
<td></td>
<td>This example would return any paragraph containing the string ‘webMethods’ followed by the new line character.</td>
</tr>
<tr>
<td>\r</td>
<td>Match a carriage return.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>doc.p[/webMethods\r/].text</td>
</tr>
<tr>
<td></td>
<td>This example would return any paragraph containing the string ‘webMethods’ followed by a carriage return.</td>
</tr>
<tr>
<td>\t</td>
<td>Match a tab character.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>doc.p[/\twebMethods/].text</td>
</tr>
<tr>
<td></td>
<td>This example would return any paragraph containing the string ‘webMethods’ preceded by a tab character.</td>
</tr>
</tbody>
</table>
**APPENDIX D  Regular Expressions**

<table>
<thead>
<tr>
<th>Use this symbol...</th>
<th>To...</th>
</tr>
</thead>
</table>
| \f                 | Match a form feed character.  
**Example** `doc.p[/webMethods\f/].text`  
This example would return any paragraph containing the string ‘webMethods’ followed by a form feed character. |
| \d                 | Match any digit. Same as [0-9].  
**Example** `doc.p[/part \d555-A/].text`  
This example would return any paragraph containing a part number that starts with any digit 0 through 9, and is followed by the characters 555-A. Therefore, it would match ‘part 1555-A’ but not ‘part A555-A’ or ‘part #555-A’. |
| \D                 | Match any non-digit. Same as [^0-9].  
**Example** `doc.p[/part \D555-A/].text`  
This example would return any paragraph containing a part number that starts with any character other than 0 through 9, and is followed by the characters 555-A. Therefore, it would match ‘part A555-A’ and ‘part #555-A’, but not ‘part 1555-A’. |
| \w                 | Match any word character. Same as [0-9a-z_A-Z].  
**Example** `doc.p[/part \w4555-A/].text`  
This example would return any paragraph containing a part number that starts with a letter or digit and is followed by the characters 555-A. Therefore, it would match ‘part A555-A’ and ‘part 1555-A’, but not ‘part #555-A’. |
| \W                 | Match any nonword character. Same as [^0-9a-z_A-Z].  
**Example** `doc.p[/part \W4555-A/].text`  
This example would return any paragraph containing a part number that starts with a character other than a letter or digit, and is followed by the characters 555-A. Therefore, it would match ‘part #555-A’ and ‘part -555-A’, but not ‘part 1555-A’ or ‘part A555-A’. |
| \s                 | Match any white-space character. Same as [\t\n\r\f].  
**Example** `doc.p[/\swebMethods/].text`  
This example would return any paragraph containing the string ‘webMethods’ if it is preceded by a tab character, a new line character, a carriage return, or a form-feed character. |
### Regular Expression Operators

<table>
<thead>
<tr>
<th>Use this symbol...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>\S</td>
<td>Match any nonwhite-space character. Same as [^\t\n\r\f].</td>
</tr>
</tbody>
</table>

**Example** `doc.p[\SwebMethods/].text`

This example would return any paragraph containing the string ‘webMethods’, if that string is not preceded by a tab character, a new line character, a carriage return, or a form-feed character.

| \0                 | Match a null string. |

**Example** `doc.p[\^\0/].text`

This example would return any paragraph that is not empty (null).

| \xnn                | Match any character with the hexadecimal value \n. |

**Example** `doc.p[\x1FwebMethods/].text`

This example would return any paragraph containing the ASCII unit-separator character (1F) followed by the characters ‘webMethods’.

| [ ]                 | Match any character within the brackets. |

**Example** `doc.p/part [023]555-A/.text`

This example would return any paragraph containing a part number that starts with the numbers 0, 2, or 3 and is followed by the characters 555-A. Therefore, it would match ‘part 0555-A’ and ‘part 2555-A’, but not ‘part 4555-A’.

The following characters have special meaning when used within brackets:

<table>
<thead>
<tr>
<th>Use this char...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>Exclude characters from the pattern.</td>
</tr>
</tbody>
</table>

**Example** `doc.p/part 4555-[^023]/.text`

This example would return any paragraph containing a part number that does not start with the numbers 0, 2, or 3, but is followed by the characters 555-A. Therefore, it would match ‘part 4555-A’ and ‘part A555-A’, but not ‘part 0555-A’.
<table>
<thead>
<tr>
<th>Use this symbol...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Specify a range of allowed characters.</td>
</tr>
</tbody>
</table>

**Example** doc.p[/part 4555-[A-M]/].text

This example would return any paragraph containing a part number that starts with any letter A through M and is followed by the characters 555-A. Therefore, it would match ‘part A555-A’ and ‘part J555-A’, but not ‘part N555-A’.
Conditional Expressions

- Overview .............................................................. 594
- Syntax ............................................................... 594
- Operators ........................................................... 597
- Precedence ......................................................... 600
- Addressing Pipeline Variables ................................. 601
Overview

SAP BC Server provides syntax and operators that you can use to create expressions for use with the BRANCH step and pipeline mapping. In a BRANCH step, you can use an expression to determine the child step that SAP BC Server executes. At run time, the first child step whose conditional expression evaluates to “true” is the one that will be executed. In pipeline mapping, you can place a condition on a map between variables. At run time, SAP BC Server only performs the map if the assigned condition evaluates to “true.”

When you write expressions, keep the following points in mind:

- Operators, variable names, and strings are case-sensitive
- White space between the tokens of an expression is ignored

For more information about the BRANCH step, see “The BRANCH Step” on page 110. For more information about applying conditions to maps between variables, see “Applying Conditions to Maps Between Variables” on page 169.

Syntax

When you create an expression, you need to determine which values to include in the expression. Values can be represented as variable names, regular expressions, numbers, and strings. The following table identifies the types of values you can use in an expression and the syntax for each value type.

<table>
<thead>
<tr>
<th>Value Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Expression</td>
<td>/regularExpression/</td>
<td>Pattern-matching string. Use the following syntax for pattern matching of variable values: variableName = /regularExpression/</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information about regular expressions, see “Regular Expressions” on page 585.</td>
</tr>
</tbody>
</table>

Example

```
sku = /^WM[0-9]+/  
```

Explanation

Evaluates to true if the `sku` variable has a value that starts with “WM” and is followed by one or more digits (WM001, WM95157).
### Syntax

<table>
<thead>
<tr>
<th>Value Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td><code>variableName</code></td>
<td>Variable name. For information about how to use this syntax to address children of other variables or elements of array variables, see “Addressing Pipeline Variables” on page 601.</td>
</tr>
<tr>
<td></td>
<td>-OR-</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>%variableName%</code></td>
<td></td>
</tr>
</tbody>
</table>

#### Example

<table>
<thead>
<tr>
<th>Explanation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>price</code></td>
<td>Value of the <code>price</code> variable.</td>
</tr>
<tr>
<td><code>%address/postalCode%</code></td>
<td>Value of the <code>postalCode</code> variable in the <code>address</code> record.</td>
</tr>
<tr>
<td><code>%poItems[0]</code></td>
<td>Value of the first element in the <code>poItems</code> array.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>&quot;string&quot;</td>
<td>Literal string. Use this value type to compare the value of a variable to a string.</td>
</tr>
<tr>
<td></td>
<td>-OR-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'string'</td>
<td></td>
</tr>
</tbody>
</table>

#### Example

<table>
<thead>
<tr>
<th>Explanation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Favorite Customer&quot;</td>
<td>Value is the literal string “Favorite Customer”</td>
</tr>
<tr>
<td>'Favorite Customer'</td>
<td>Value is the literal string “Favorite Customer”</td>
</tr>
</tbody>
</table>

**Note:** Strings not enclosed in quotes (‘ or ”) are interpreted as variable names.

<table>
<thead>
<tr>
<th>Value Type</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td><code>number</code></td>
<td>Number. The following examples indicate the accepted number formats:</td>
</tr>
</tbody>
</table>

#### Examples

<table>
<thead>
<tr>
<th>Explanation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-10, 5, 100</code></td>
<td>Integers</td>
</tr>
<tr>
<td><code>5.0, 6.02</code></td>
<td>Floating point number (java.lang.Double)</td>
</tr>
<tr>
<td><code>6.345e+4</code></td>
<td>Scientific notation</td>
</tr>
</tbody>
</table>
### Checking for Variable Existence

Sometimes you might want to create an expression that checks only for the existence of a variable in the pipeline or checks to see whether a variable is null. The following table describes the syntax used to check the pipeline for variable existence.

<table>
<thead>
<tr>
<th>To check for</th>
<th>Use this syntax</th>
<th>Description</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Existence</td>
<td><code>variableName</code></td>
<td>Evaluates to true if the specified variable exists in the pipeline and has a non-null value.</td>
<td><code>customerID</code></td>
<td>Evaluates to true if the <code>customerID</code> variable exists in the pipeline and is not null.</td>
</tr>
<tr>
<td>Variable Does Not Exist</td>
<td><code>!variableName</code></td>
<td>Evaluates to true if the specified variable does not exist in the pipeline or is null.</td>
<td><code>!quantity</code></td>
<td>Evaluates to true if the <code>quantity</code> variable does not exist in the pipeline or is null.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>!color &amp; !size</code></td>
<td>Evaluates to true if the <code>color</code> variable does not exist in the pipeline or is null and the <code>size</code> variable does not exist in the pipeline or is null.</td>
</tr>
</tbody>
</table>
Operators

Expressions can include relational and logical operators. Relational operators are used to compare values to each other. Logical operators are used to combine multiple expressions into a single condition.

Relational Operators

You can use the following relational operators in expressions:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Syntax</th>
<th>Description</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>$a = b$</td>
<td>Equal to.</td>
<td>$\text{customerID} = \text{&quot;webMethods&quot;}$</td>
<td>Evaluates to true if the value of the $\text{customerID}$ variable is \text{&quot;webMethods&quot;}.</td>
</tr>
<tr>
<td>==</td>
<td>$a == b$</td>
<td>Equal to.</td>
<td>$\text{sku} = \text{&quot;WM001&quot;}$</td>
<td>Evaluates to true if the value of the $\text{sku}$ variable is \text{&quot;WM001&quot;}.</td>
</tr>
<tr>
<td>!=</td>
<td>$a != b$</td>
<td>Not equal to.</td>
<td>$\text{quantity} != 0$</td>
<td>Evaluates to true if the value of the $\text{quantity}$ variable does not equal \text{0} (zero).</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>$a &lt;&gt; b$</td>
<td>Not equal to.</td>
<td>$\text{state} &lt;&gt; \text{&quot;ME&quot;}$</td>
<td>Evaluates to true if the value of the $\text{state}$ variable does not equal ME (Maine).</td>
</tr>
<tr>
<td>&gt;</td>
<td>$a &gt; b$</td>
<td>Greater than.</td>
<td>$\text{price} &gt; 100$</td>
<td>Evaluates to true if the value of the $\text{price}$ variable is greater than \text{100}.</td>
</tr>
</tbody>
</table>

| %companyID% > \text{"Acme"} | Evaluates to true if the value of the $\text{companyID}$ variable is alphabetically greater than Acme. |
Note: When using relational operators to compare strings, SAP BC Server considers A to be the lowest letter and Z to be the highest. (e.g. A < B, A < Z).

<table>
<thead>
<tr>
<th>Operator</th>
<th>Syntax</th>
<th>Description</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&gt;=</code></td>
<td><code>a &gt;= b</code></td>
<td>Greater than or equal to.</td>
<td>%totalPrice% &gt;= 100</td>
<td>Evaluates to true if the value of the $totalPrice$ variable is greater than or equal to 100.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>companyID &gt;= &quot;Acme&quot;</td>
<td>Evaluates to true if the value of the $companyID$ variable is alphabetically greater than or equal to Acme.</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td><code>a &lt; b</code></td>
<td>Less than.</td>
<td>quantity &lt; 5</td>
<td>Evaluates to true if the value of the $quantity$ variable is less than 5.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>companyID &lt; &quot;Acme&quot;</td>
<td>Evaluates to true if the value of the $companyID$ variable is alphabetically less than Acme.</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td><code>a &lt;= b</code></td>
<td>Less than or equal to.</td>
<td>unitPrice &lt;= 100</td>
<td>Evaluates to true if the value of the $unitPrice$ variable is less than or equal to 100.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>companyID &lt;= &quot;Acme&quot;</td>
<td>Evaluates to true if the value of the $companyID$ variable is alphabetically less than or equal to Acme.</td>
</tr>
</tbody>
</table>
Logical Operators

You can use the following logical operators in expressions to create conditions consisting of more than one expression:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>! expr</td>
<td>Negates the next expression.</td>
</tr>
<tr>
<td>not</td>
<td>not expr</td>
<td>Negates the next expression.</td>
</tr>
<tr>
<td></td>
<td>expr</td>
<td>Logical OR. True if either of the expressions are true.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>expr</td>
</tr>
<tr>
<td>or</td>
<td>expr or expr</td>
<td>Logical OR. True if either of the expressions are true.</td>
</tr>
</tbody>
</table>

**Example**

**Explanation**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>! (%SKU% = &quot;WM001&quot;)</td>
<td>Evaluates to true if the value of the SKU variable is not equal to WM001.</td>
</tr>
<tr>
<td>not (color = &quot;blue&quot;)</td>
<td>Evaluates to true if the color variable is not equal to blue.</td>
</tr>
<tr>
<td>%COLOR% = &quot;blue&quot;</td>
<td>Evaluates to true if the value of the color variable is blue.</td>
</tr>
<tr>
<td>totalPrice &gt; 1000</td>
<td></td>
</tr>
<tr>
<td>creditCardNum = $null or cardExpireDate = $null or cardExpireDate &lt;= orderDate</td>
<td>Evaluates to true if the value of the creditCardNum variable is null or missing or if the value of the cardExpireDate variable is null or missing or if the value of the cardExpireDate variable is less than or equal to the value of the orderDate variable.</td>
</tr>
</tbody>
</table>
APPENDIX E  Conditional Expressions

<table>
<thead>
<tr>
<th>Operator</th>
<th>Syntax</th>
<th>Description</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>expr &amp; expr</td>
<td>Logical AND. Both expressions must evaluate to true for the entire condition to be true.</td>
<td>%customerID% = 'Favorite Customer' &amp; %sku% = 'WM001'</td>
<td>Evaluates to true if the value of the customerID variable is Favorite Customer and the value of the sku variable is WM001.</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>expr &amp;&amp; expr</td>
<td>Logical AND. Both expressions must evaluate to true for the entire condition to be true.</td>
<td>quantity &gt;= 20 &amp;&amp; totalPrice &gt;= 100</td>
<td>Evaluates to true if the value of the quantity variable is greater than or equal to 20 and the value of the totalPrice variable is greater than or equal to 100.</td>
</tr>
<tr>
<td>and</td>
<td>expr and expr</td>
<td>Logical AND. Both expressions must evaluate to true for the entire condition to be true.</td>
<td>!color and !size</td>
<td>Evaluates to true if the color variable does not exist in the input or is null and the size variable does not exist in the input or is null.</td>
</tr>
</tbody>
</table>

**Precedence**

SAP BC Server evaluates expressions in a condition according to the precedence level of the operators in the expressions.

The following table identifies the precedence level of each operator you can use in an expression.

<table>
<thead>
<tr>
<th>Precedence Level</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>()</td>
</tr>
<tr>
<td>2</td>
<td>not, !</td>
</tr>
<tr>
<td>3</td>
<td>=, ==, !=, &lt;=, &lt;, &gt;, &gt;=, &lt;, &lt;=</td>
</tr>
</tbody>
</table>
Addressing Pipeline Variables

4 and, & &
5 or, | | |

**Note:** To override the order in which expressions in a condition are evaluated, enclose the operations you want evaluated first in parentheses. SAP BC Server evaluates expressions contained in parentheses first.

**Note:** When using relational operators to compare strings, SAP BC Server considers A to be the lowest letter and Z to be the highest. (e.g., A < B, A < Z).

**Addressing Pipeline Variables**

In an expression, you can refer to the values of variables that are children of other variables and refer to the values of elements in an array variable. To address children of variables or an element in an array, you need to use a directory-like notation to describe the position of the value.

**Use this notation...**

variableName

address a variable in the pipeline.

*Example:* state

Variable *state*.

variableName/childVariableName

address the child variable of a variable in the pipeline.

*Example:* %buyerInfo/state%

Variable *state* within Record *buyerInfo*.

arrayVariableName[index]

address an element in an array.

*Example:* orderItems[0]

Value of the first element in the *orderItems* array.

arrayVariableName[rowIndex][columnIndex]

address an element in a two-dimensional array (String Table).

*Example:* dictionary[1][2]

Value of the element located in the third column of the second row in the *dictionary* array.
Note: You can enclose variable names in %, for example %buyerInfo/state%.

The following illustration shows how the examples in the above table would appear in the pipeline.

Pipeline tab for examples in the above table

For more information about using variables as values in expressions, see “Syntax” on page 594.
jcode tags

- jcode Template ................................................................. 604
- jcode Examples ................................................................. 605
jcode Template

The following code provides a template describing the tags (highlighted) that the jcode utility uses to identify code segments in a Java source file. The examples explain in detail how to specify different service properties within your Java code so that it is only necessary to store the Java source files in a revision system.

```java
package Folder0;
/**
 * This is an example of an empty Java source code file,
 * properly annotated for use with the jcode utility. The correct
 * annotation is extremely important for services developed on
 * versions before 4.8 that should be migrated with the jcode
 * utility.
 */
import com.wm.app.b2b.server.Service;
import com.wm.app.b2b.server.ServiceException;
import com.wm.data.*;
import com.wm.util.Values;
// --- <<B2B-START-IMPORTS>> ---
// --- <<B2B-END-IMPORTS>> ---

public class Folder1
{
    public static void Service1 (IData pipeline)
        throws ServiceException
    {
        // --- <<B2B-START(Service1)>> ---
        // --- <<B2B-END>> ---
        return;
    }

    public static void Service2 (IData pipeline)
        throws ServiceException
    {
        // --- <<B2B-START(Service2)>> ---
        // --- <<B2B-END>> ---
        return;
    }

    // --- <<B2B-START-SHARED>> ---
    // --- <<B2B-END-SHARED>> ---
}
```
jcode Examples

The following are complete examples of properly commented Java source code. Check them carefully to be aware of all options you have to provide information for services within your Java coding using the jcode annotations.

Sample Code - IData

The following is an example of a class whose services (methods) take IData objects as input.

```java
package recording;
/**
 * This is an example of Java source code properly annotated for use with the SAP BC jcode utility. Note that, unless noted otherwise, all comments will be stripped out of this file during the process of fragmenting the code for further editing with the SAP BC Developer...
 */
/**
 * == IMPORTS ==
 * All your imports should be wrapped with the START-IMPORTS and END-IMPORTS tags.
 */
import com.wm.app.b2b.server.Service;
import com.wm.app.b2b.server.ServiceException;
import com.wm.data.*;
import com.wm.util.Values;
// --- <<B2B-START-IMPORTS>> ---
import java.util.*;
// --- <<B2B-END-IMPORTS>> ---
/**
 * == CLASS NAMING ==
 * This class contains the definition of all the Java services within the recording.accounts folder (note the recording package declaration up top). Note that each service is defined by a method with the same name.
 * As of Business Connector 4.8 you can also prefix your class with JSBC_ and thus make sure to avoid class/package name clashes from the beginning, e.g. in the example the class would be named JSBC_accounts, which would still lead to a service located in the folder recording.accounts after a jcode frag.
 */
public class accounts {
    /**
     * == INDIVIDUAL SERVICES ==
     * The createAccount service. This service expects three parameters -- a string ("name"), a string array ("references"), and a record. It returns two strings ("message" and "id").
     * Note the special tags delimiting the start and end of the service. The two lines immediately before start tag and after the end tags are mandatory.
     */
```
* Also note the use of comments to establish a signature for the
* service. Each signature line has the following format:
* ['['direction']' type:dimension:option name [recordname] ['{'picklist'}']
* ['"''] [''#'comment]
* direction: "i" (input) or "o" (output)
* type:
* - field (corresponds to instances of java.lang.String)
* - fieldpw a field for which the '*' sign is shown, when it is typed, e.g.
* passwords
* - record (corresponds to instances of com.wm.data.IData, i.e. a record)
* - recref (corresponds to a record reference, i.e. points to an
* existing definition in the namespace)
* - object (corresponds to instances of any other class)
* dimension:
* - 0: a single value
* - 1. an array of values, e.g. a record list or a string list
* - 2. a 2-dimensional table of values, i.e. string table

* option:
* - required (this parameter is mandatory)
* - optional (this parameter is optional)
* name: the name of the parameter*
* recordname:
* Required parameter if and only if type is recref. It defines the
* referenced record for this parameter in the namespace,
* e.g. pub.flow:transportInfo.
* picklist:
* a comma separated list of values that are shown in a select box,
* when executing the service. If succeeded by '*', it is possible to
* insert a different value as well, i.e. the picklist is editable
* comment:
* if there is a '#' sign all the rest of the line is taken as
* comment for the signature field, which is shown in the properties dialog
* To indicate nesting, use a single "-" at the beginning of
* each line for each level of nesting.

public static void createAccount (IData pipeline)
    throws ServiceException
{    // --- <<B2B-START(createAccount)>> ---
    // @sigtype java 3.5
    // [i] field:0:required name # name of the person
    // [i] field:0:required gender {male, female}
    // [i] field:1:required references
    // [i] recref:0:required customer recording.accounts:customerInfo
    // [i] record:0:required data
    // [i] - field:1:required address
    // [i] - field:1:required phone
    // [i] - fieldpw:0:required pin
    // [o] field:1:required message
    // [o] field:1:required id
    IDataCursor idc = pipeline.getCursor();
    idc.first("name");
String name = IDataUtil.getString(idc);
idc.first("references");
String [] refs = IDataUtil.getStringArray(idc);
idc.first("data");
IData data = IDataUtil.getIData(idc);

// Do something with the information here. Note that this
// comment inside the service body is the only one that won't
// get discarded when fragmenting the service (i.e., it will
// show up in Developer.)
IDataUtil.put
(idc, "message", "createAccount not fully implemented");
IDataUtil.put (idc, "id", "00000000");
idc.destroy();
// --- <<B2B-END>> ---
return;
}

/**
 * There are some other special comment lines, that can
 * precede the signature fields or can make them unnecessary
 * @sigtype signature version
 * Specifies which method style the Java Service uses
 * - java 3.0
 * Stands for the classic Values service methods
 * - java 3.5
 * Stands for the IData service method
 * @spec specification
 * Allows referring to a specification defined in the namespace
 * instead of specifying the signature with the signature comments.
 * The specification is identified by its namespace name.
 * @deprecated deprecation comment
 * This annotation allows marking a service as deprecated.
 * The deprecation comment will be shown in the Developer at several
 * places.
 * @stateless
 * Marks the service to be stateless
 */
public static void createSingleAccount (IData pipeline)
throws ServiceException
{
// --- <<B2B-START(createSingleAccount)>> ---
// @sigtype java 3.5
// @spec recording:singleAccountSignature
// @deprecated use recording.account:createAccount instead
IDataCursor idc = pipeline.getCursor();
String name = IDataUtil.getString(idc, "name");
IData data = IDataUtil.getIData(idc, "data");
// Do something with the information here.
IDataUtil.put(idc, "message", "createSingleAccount not fully implemented");
IDataUtil.put(idc, "id", "00000000");
idc.destroy();
// --- <<B2B-END>> ---
return;
}
/**
 * == COMPLEX SIGNATURES ==
 * The getAccount service. This service takes a single string
 * "id", and returns a complex structure representing the
 * account information. Note the use of the helper functions
 * (defined below).
 */
public static void getAccount (IData pipeline)
throws ServiceException
{
    // --- <<B2B-START(getAccount)>> ---
    // @sigtype java 3.5
    // [i] field:0:required id
    // [o] record:1:required account
    // [o] - field:0:required name
    // [o] - field:1:required refs
    // [o] - record:0:required contact
    // [o] -- field:0:required address
    // [o] -- field:0:required phone
    // --- <<BC-COMMENT>> ---
    // Line comments between the enclosing two annotation comments
    // will appear in the comments text area on the Input/Output tab
    // in the Developer and can be used for documenting services.
    // --- <<BC-COMMENT-END>> ---
    IDataCursor idc = pipeline.getCursor();
    if(idc.first("id"))
    {
        try
        {
            String id = IDataUtil.getString(idc);
            IData data = getAccountInformation(id);
            idc.last();
            idc.insertAfter ("account", data);
        }
        catch (Exception e)
        {
            throw new ServiceException(e.toString());
        }
    }
    idc.destroy();
    // --- <<B2B-END>> ---
    return;
}
/**
 * == SHARED SOURCE ==
 * This is where the shared code lives. This includes both
 * global data structures and non-public functions that aren't
 * exposed as Services. Note the tags delimiting the start
 * and end of the shared code section.
 */
// --- <<B2B-START-SHARED>> ---
private static Vector accounts = new Vector();
private static IData getAccountInformation (String id) {
    throw new RuntimeException ("this service is not implemented yet");
}
Sample Code - Values

The following is an example of a class whose services (methods) take Values objects as input. (Note that Values has been replaced by the IData object. For all new code development, we recommend that you use IData, instead of Values.)

```java
public class accounts {
    public static Values createAccount (Values in) {
        Values out = in;
        String name = in.getString("name");
        String [] refs = in.getStringArray("references");
        Values data = in.getValues("data");
        String address = in.getString("address");
        String phone = in.getString("phone");
        String pin = in.getString("pin");
        String message = in.getString("message");
        String id = in.getString("id");

        // Do something with the information here. Note that this
        // comment inside the service body is the only one that won't
        // get discarded when fragmenting the service (i.e., it will
        // show up in Developer.)
        out.put ("message", "createAccount not fully implemented");
        out.put ("id", "00000000");
        return out;
    }

    public static Values createSingleAccount (Values in) {
        Values out = in;
        return out;
    }
}
```
public static Values getAccount (Values in) {
    Values out = in;
    // --- <<B2B-START(getAccount)>> ---
    // @sigtype java 3.0
    // [i] field:0:required id
    // [o] record:1:required account
    // [o] - field:0:required name
    // [o] - field:1:required refs
    // [o] - record:0:required contact
    // [o] -- field:0:required address
    // [o] -- field:0:required phone
    String id = in.getString("id");
    Values data = getAccountInformation(id);
    out.put("account", data);
    // --- <<B2B-END>> ---
    return out;
}

// --- <<B2B-START-SHARED>> ---
private static Vector accounts = new Vector();
private static Values getAccountInformation (String id) {
    throw new RuntimeException ("this service is not implemented yet");
}
// --- <<B2B-END-SHARED>> ---
Validation Content Constraints

- Overview ................................................................. 612
- Content Types ......................................................... 612
- Constraining Facets .................................................. 623
Overview

You can apply content constraints to variables in the records, specifications, or service signatures that you want to use as blueprints in data validation. Content constraints describe the data a variable can contain. At validation time, if the variable value does not conform to the content constraints applied to the variable, the validation engine considers the value to be invalid. For more information about validation, see “Performing Data Validation” on page 209.

When applying content constraints to variables, you can do the following:

- **Select a content type.** A content type specifies the type of data for the variable value, such as string, integer, boolean, and date. A content type corresponds to a simple type definition in a schema.

- **Set constraining facets.** Constraining facets restrict the content type, which in turn, restrict the value of the variable to which the content type is applied. Each content type has a set of constraining facets. For example, you can set a length restriction for a string content type, or a maximum value restriction for an integer content type.

For example, for a String variable named `itemQuantity`, you might specify a content type that requires the variable value to be an integer. You could then set constraining facets that limit the content of `itemQuantity` to a value between 1 and 100.

The content types and constraining facets described in this appendix correspond to the built-in data types and constraining facets in XML Schema. The World Wide Web Consortium (W3C) defines the built-in data types and constraining facets in the specification *XML Schema Part 2: Datatypes* ([http://www.w3c.org/TR/xmlschema-2](http://www.w3c.org/TR/xmlschema-2)).

Content Types

The following table identifies the content types you can apply to String, String List, or String Table variables. Each of these content types corresponds to a built-in simple type defined in the specification *XML Schema Part 2: Datatypes*.

<table>
<thead>
<tr>
<th>Content Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anyURI</td>
<td>A Uniform Resource Identifier Reference. The value of anyURI may be absolute or relative.</td>
</tr>
</tbody>
</table>

**Constraining Facets**

- enumeration, length, maxLength, minLength, pattern

**Note:** The anyURI type indicates that the variable value plays the role of a URI and is defined like a URI. SAP BC Server does not validate URI references because it is impractical for applications to check the validity of a URI reference.
### Content Types

<table>
<thead>
<tr>
<th>Content Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>base64Binary</strong></td>
<td>Base64-encoded binary data.</td>
</tr>
<tr>
<td><strong>boolean</strong></td>
<td>True or false.</td>
</tr>
<tr>
<td><strong>byte</strong></td>
<td>A whole number whose value is greater than or equal to -128 but less than or equal to 127.</td>
</tr>
<tr>
<td><strong>date</strong></td>
<td>A calendar date from the Gregorian calendar. Values need to match the following pattern: CCYY-MM-DD</td>
</tr>
</tbody>
</table>

Where CC represents the century, YY the year, MM the month, DD the day. The pattern can include a Z at the end to indicate Coordinated Universal Time or to indicate the difference between the time zone and coordinated universal time.

| **Constraining Facets** | enumeration, length, maxLength, minLength, pattern |
| **Constraining Facets** | pattern |
| **Example** | true, 1, false, 0 |
| **Example** | -128, -26, 0, 15, 125 |
| **Example** | 1997-08-09 (August 9, 1997) |

**Constraining Facets**
- enumeration
- fractionDigits
- maxExclusive
- maxInclusive
- minExclusive
- minInclusive
- pattern
- totalDigits
## DateTime

A specific instant of time (a date and time of day). Values need to match the following pattern:

\[
CCYY-MM-DDThh:mm:ss.sss
\]

Where \(CC\) represents the century, \(YY\) the year, \(MM\) the month, \(DD\) the day, \(T\) the date/time separator, \(hh\) the hour, \(mm\) the minutes, and \(ss\) the seconds. The pattern can include a \(Z\) at the end to indicate Coordinated Universal Time or to indicate the difference between the time zone and coordinated universal time.

**Constraining Facets**

- enumeration, maxExclusive, minInclusive, minInclusive, pattern

**Example**

2000-06-29T17:30:00-05:00 represents 5:30 pm Eastern Standard time on June 29, 2000. (Eastern Standard Time is 5 hours behind Coordinated Universal Time.)

## Decimal

A number with an optional decimal point.

**Constraining Facets**

- enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits

**Example**

8.01, 290, -47.24

## Double

Double-precision 64-bit floating point type.

**Constraining Facets**

- enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern

**Example**

6.02E23, 3.14, -26, 1.25e-2
### Content Types

<table>
<thead>
<tr>
<th>Content Types</th>
<th>Description</th>
</tr>
</thead>
</table>
| **duration**  | A length of time. Values need to match the following pattern:  
\[ P^Y nM nD T^h nH nM nS \]  
Where \( nY \) represents the number of years, \( nM \) the number of months, \( nD \) the number of days, \( T \) separates the date and time, \( nH \) the number of hours, \( nM \) the number of minutes and \( nS \) the number of seconds. Precede the duration with a minus (-) sign to indicate a negative duration.  
**Constraining Facets**  
enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern  
**Example**  
`P2Y10M20DT5H50M` represents a duration of 2 years, 10 months, 20 days, 5 hours, and 50 minutes |
| **ENTITIES**  | Sequence of whitespace-separated ENTITY values declared in the DTD. Represents the ENTITIES attribute type from the XML 1.0 Recommendation.  
**Constraining Facets**  
enumeration, length, maxLength, minLength |
| **ENTITY**    | Name associated with an unparsed entity of the DTD. Represents the ENTITY attribute type from the XML 1.0 Recommendation.  
**Constraining Facets**  
enumeration, length, maxLength, minLength, pattern, whiteSpace |
| **float**      | A number with a fractional part.  
**Constraining Facets**  
enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern  
**Example**  
8.01, 25, 6.02E23, -5.5 |
<table>
<thead>
<tr>
<th>Content Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gDay</td>
<td>A specific day that recurs every month. Values need to match the following pattern:</td>
</tr>
<tr>
<td></td>
<td>--DD</td>
</tr>
<tr>
<td></td>
<td>Where DD represents the day. The pattern can include a Z at the end to indicate Coordinated</td>
</tr>
<tr>
<td></td>
<td>Universal Time or to indicate the difference between the time zone and coordinated universal</td>
</tr>
<tr>
<td></td>
<td>time.</td>
</tr>
<tr>
<td></td>
<td><strong>Constraining Facets</strong></td>
</tr>
<tr>
<td></td>
<td>enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong></td>
</tr>
<tr>
<td></td>
<td>--24 indicates the 24th of each month</td>
</tr>
<tr>
<td>gMonth</td>
<td>A Gregorian month that occurs every year. Values need to match the following pattern:</td>
</tr>
<tr>
<td></td>
<td>--MM</td>
</tr>
<tr>
<td></td>
<td>Where MM represents the month. The pattern can include a Z at the end to indicate Coordinated</td>
</tr>
<tr>
<td></td>
<td>Universal Time or to indicate the difference between the time zone and coordinated universal</td>
</tr>
<tr>
<td></td>
<td>time.</td>
</tr>
<tr>
<td></td>
<td><strong>Constraining Facets</strong></td>
</tr>
<tr>
<td></td>
<td>enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong></td>
</tr>
<tr>
<td></td>
<td>--11 represents November</td>
</tr>
<tr>
<td>gMonthDay</td>
<td>A specific day and month that recurs every year in the Gregorian calendar. Values need to match</td>
</tr>
<tr>
<td></td>
<td>the following pattern:</td>
</tr>
<tr>
<td></td>
<td>--MM-DD</td>
</tr>
<tr>
<td></td>
<td>Where MM represents the month and DD represents the day. The pattern can include a Z at the</td>
</tr>
<tr>
<td></td>
<td>end to indicate Coordinated Universal Time or to indicate the difference between the time zone</td>
</tr>
<tr>
<td></td>
<td>and coordinated universal time.</td>
</tr>
<tr>
<td></td>
<td><strong>Constraining Facets</strong></td>
</tr>
<tr>
<td></td>
<td>enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong></td>
</tr>
<tr>
<td></td>
<td>--09-24 represents September 24th</td>
</tr>
</tbody>
</table>
## Content Types

<table>
<thead>
<tr>
<th>Content Types</th>
<th>Description</th>
</tr>
</thead>
</table>
| **gYear** | A specific year in the Gregorian calendar. Values need to match the following pattern:  

```
CCYY
```

Where CC represents the century, and YY the year. The pattern can include a Z at the end to indicate Coordinated Universal Time or to indicate the difference between the time zone and coordinated universal time.

**Constraining Facets**  
enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern  

**Example**  
2001 indicates 2001 |
| **gYearMonth** | A specific month and year in the Gregorian calendar. Values need to match the following pattern:  

```
CCYY-MM
```

Where CC represents the century, YY the year, and MM the month. The pattern can include a Z at the end to indicate Coordinated Universal Time or to indicate the difference between the time zone and coordinated universal time.

**Constraining Facets**  
enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern  

**Example**  
2001-04 indicates April 2001 |
| **hexBinary** | Hex-encoded binary data.  

**Constraining Facets**  
enumeration, length, maxLength, minLength, pattern |
| **ID** | A name that uniquely identifies an individual element in an instance document. The value for ID needs to be a valid XML name. The ID datatype represents the ID attribute type from the XML 1.0 Recommendation.  

**Constraining Facets**  
enumeration, length, maxLength, minLength, pattern, whiteSpace |
### Content Types

<table>
<thead>
<tr>
<th>Content Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IDREF</strong></td>
<td>A reference to an element with a unique ID. The value of IDREF is the same as the ID value. The IDREF datatype represents the IDREF attribute type from the XML 1.0 Recommendation.</td>
</tr>
<tr>
<td></td>
<td><strong>Constraining Facets</strong></td>
</tr>
<tr>
<td></td>
<td>enumeration, length, maxLength, minLength, pattern, whiteSpace</td>
</tr>
<tr>
<td><strong>IDREFS</strong></td>
<td>Sequence of white space separated IDREFs used in an XML document. The IDREFS datatype represents the IDREFS attribute type from the XML 1.0 Recommendation.</td>
</tr>
<tr>
<td></td>
<td><strong>Constraining Facets</strong></td>
</tr>
<tr>
<td></td>
<td>enumeration, length, maxLength, minLength</td>
</tr>
<tr>
<td><strong>int</strong></td>
<td>A whole number with a value greater than or equal to -2147483647 but less than or equal to 2147483647.</td>
</tr>
<tr>
<td></td>
<td><strong>Constraining Facets</strong></td>
</tr>
<tr>
<td></td>
<td>enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong></td>
</tr>
<tr>
<td></td>
<td>-21474836, -55500, 0, 33123, 4271974</td>
</tr>
<tr>
<td><strong>integer</strong></td>
<td>A positive or negative whole number.</td>
</tr>
<tr>
<td></td>
<td><strong>Constraining Facets</strong></td>
</tr>
<tr>
<td></td>
<td>enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong></td>
</tr>
<tr>
<td></td>
<td>-2500, -5, 0, 15, 365</td>
</tr>
<tr>
<td><strong>language</strong></td>
<td>Language identifiers used to indicate the language in which the content is written. Natural language identifiers are defined in IETF RFC 1766.</td>
</tr>
<tr>
<td></td>
<td><strong>Constraining Facets</strong></td>
</tr>
<tr>
<td></td>
<td>enumeration, length, maxLength, minLength, pattern, whiteSpace</td>
</tr>
<tr>
<td><strong>long</strong></td>
<td>A whole number with a value greater than or equal to -9223372036854775808 but less than or equal to 9223372036854775807.</td>
</tr>
<tr>
<td></td>
<td><strong>Constraining Facets</strong></td>
</tr>
<tr>
<td></td>
<td>enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong></td>
</tr>
<tr>
<td></td>
<td>-5560, -23, 0, 256, 3211569432</td>
</tr>
<tr>
<td>Content Types</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Name</td>
<td>XML names that match the Name production of XML 1.0 (Second Edition).</td>
</tr>
<tr>
<td></td>
<td><strong>Constraining Facets</strong></td>
</tr>
<tr>
<td></td>
<td>enumeration, length, maxLength, minLength, pattern, whiteSpace</td>
</tr>
<tr>
<td>NCName</td>
<td>Non-colonized XML names. Set of all strings that match the NCName production of Namespaces in XML.</td>
</tr>
<tr>
<td></td>
<td><strong>Constraining Facets</strong></td>
</tr>
<tr>
<td></td>
<td>enumeration, length, maxLength, minLength, pattern, whiteSpace</td>
</tr>
<tr>
<td>negativeInteger</td>
<td>An integer with a value less than or equal to −1.</td>
</tr>
<tr>
<td></td>
<td><strong>Constraining Facets</strong></td>
</tr>
<tr>
<td></td>
<td>enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong></td>
</tr>
<tr>
<td></td>
<td>−255556, −354, −3, −1</td>
</tr>
<tr>
<td>NM_TOKEN</td>
<td>Any mixture of name characters. Represents the NM_TOKEN attribute type from the XML 1.0 Recommendation.</td>
</tr>
<tr>
<td></td>
<td><strong>Constraining Facets</strong></td>
</tr>
<tr>
<td></td>
<td>enumeration, length, maxLength, minLength, pattern, whiteSpace</td>
</tr>
<tr>
<td>NM_TOKENS</td>
<td>Sequences of NM_TOKENS. Represents the NM_TOKENS attribute type from the XML 1.0 Recommendation.</td>
</tr>
<tr>
<td></td>
<td><strong>Constraining Facets</strong></td>
</tr>
<tr>
<td></td>
<td>enumeration, length, maxLength, minLength</td>
</tr>
<tr>
<td>nonNegativeInteger</td>
<td>An integer with a value greater than or equal to 0.</td>
</tr>
<tr>
<td></td>
<td><strong>Constraining Facets</strong></td>
</tr>
<tr>
<td></td>
<td>enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong></td>
</tr>
<tr>
<td></td>
<td>0, 15, 32123</td>
</tr>
</tbody>
</table>
## APPENDIX G Validation Content Constraints

<table>
<thead>
<tr>
<th>Content Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nonPositiveInteger</code></td>
<td>An integer with a value less than or equal to 0.</td>
</tr>
<tr>
<td><strong>Constraining Facets</strong></td>
<td>enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits, whiteSpace</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td><code>-256453, -357, -1, 0</code></td>
</tr>
<tr>
<td><code>normalizedString</code></td>
<td>Represents white space normalized strings. Set of strings (sequence of UCS characters) that do not contain the carriage return (<code>\x1b</code>), line feed (<code>\x0a</code>), or tab (<code>\x09</code>) characters.</td>
</tr>
<tr>
<td><strong>Constraining Facets</strong></td>
<td>enumeration, length, maxLength, minLength, pattern, whiteSpace</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td><code>MAB-0907</code></td>
</tr>
<tr>
<td><code>positiveInteger</code></td>
<td>An integer with a value greater than or equal to 1.</td>
</tr>
<tr>
<td><strong>Constraining Facets</strong></td>
<td>enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td><code>1, 1500, 23000</code></td>
</tr>
<tr>
<td><code>short</code></td>
<td>A whole number with a value greater than or equal to -32768 but less than or equal to 32767.</td>
</tr>
<tr>
<td><strong>Constraining Facets</strong></td>
<td>enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td><code>-32000, -543, 0, 456, 3265</code></td>
</tr>
<tr>
<td><code>string</code></td>
<td>Character strings in XML. A sequence of UCS characters (ISO 10646 and Unicode). By default, all white space is preserved for variables with a string content constraint.</td>
</tr>
<tr>
<td><strong>Constraining Facets</strong></td>
<td>enumeration, length, maxLength, minLength, pattern, whiteSpace</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td><code>MAB-0907</code></td>
</tr>
<tr>
<td>Content Types</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| time          | An instant of time that occurs every day. Values need to match the following pattern:  
|               | hh:mm:ss.sss  
|               | Where hh indicates the hour, mm the minutes, and ss the seconds. The pattern can include a Z at the end to indicate Coordinated Universal Time or to indicate the difference between the time zone and coordinated universal time.  
|               | **Constraining Facets**  
|               | enumeration, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern  
|               | **Example**  
|               | 18:10:00-05:00 (6:10 pm, Eastern Standard Time) Eastern Standard Time is 5 hours behind Coordinated Universal Time. |
| token         | Represents tokenized strings. Set of strings that do not contain the carriage return (#xD), line feed (#xA), or tab (#x9) characters, leading or trailing spaces (#x20), or sequences of two or more spaces.  
|               | **Constraining Facets**  
|               | enumeration, length, maxLength, minLength, pattern, whiteSpace  
| unsignedByte  | A whole number greater than or equal to 0, but less than or equal to 255.  
|               | **Constraining Facets**  
|               | enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits  
|               | **Example**  
|               | 0, 112, 200  
| unsignedInt   | A whole number greater than or equal to 0, but less than or equal to 4294967295.  
|               | **Constraining Facets**  
|               | enumeration, fractionDigits, maxExclusive, maxInclusive, minExclusive, minInclusive, pattern, totalDigits  
|               | **Example**  
|               | 0, 22335, 123223333  

SAP BC Developer Guide 4.8
### Content Types

<table>
<thead>
<tr>
<th>Content Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>unsignedLong</code></td>
<td>A whole number greater than or equal to 0, but less than or equal to 18446744073709551615.</td>
</tr>
<tr>
<td><strong>Constraining Facets</strong></td>
<td><code>enumeration</code>, <code>fractionDigits</code>, <code>maxExclusive</code>, <code>maxInclusive</code>, <code>minExclusive</code>, <code>minInclusive</code>, <code>pattern</code>, <code>totalDigits</code></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>0, 2001, 3363124</td>
</tr>
<tr>
<td><code>unsignedShort</code></td>
<td>A whole number greater than or equal to 0, but less than or equal to 65535.</td>
</tr>
<tr>
<td><strong>Constraining Facets</strong></td>
<td><code>enumeration</code>, <code>fractionDigits</code>, <code>maxExclusive</code>, <code>maxInclusive</code>, <code>minExclusive</code>, <code>minInclusive</code>, <code>pattern</code>, <code>totalDigits</code></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>0, 1000, 65000</td>
</tr>
</tbody>
</table>

**Note:** In earlier versions of SAP BC Server (SAP BC versions 3.5 through version 4.0), you could apply the NOTATION and QName content types to variables. However, these implementations were based on earlier drafts of XML Schema specification produced by the W3C. In the XML Schema Recommendation, the definition of NOTATION and QName changed. These types can only be used as the type definitions for elements or attributes in an XML Schema definition used to validate an XML instance document. Consequently, you can no longer apply the NOTATION and QName content types to variables in a record, a specification, or to a service signature.
Constraining Facets

When you apply a content type to a variable, you can also set constraining facets for the content type. Constraining facets are properties that further define the content type. For example, you can set a minimum value or precision value for a decimal content type. Each content type has a set of constraining facets. The constraining facets described in the following table correspond to constraining facets defined in the specification XML Schema Part 2: Datatypes.

<table>
<thead>
<tr>
<th>Constraining Facet</th>
<th>Description</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>enumeration</td>
<td>The possible values for the variable at run time.</td>
<td>If you also entered possible values using the Pick List feature on the General tab, those values will be displayed at run time. However, the enumeration values will be used for validation.</td>
</tr>
<tr>
<td>fractionDigits</td>
<td>The maximum number of digits to the right of the decimal point. For example, the fractionDigits of the value 999.99 is 2.</td>
<td>fractionDigits needs to be less than or equal to totalDigits.</td>
</tr>
<tr>
<td>length</td>
<td>The precise units of length required for the variable value.</td>
<td>If you specify length, you cannot specify either minLength or maxLength.</td>
</tr>
<tr>
<td>maxExclusive</td>
<td>The upper bound of a range of possible values. The range excludes the value you specify. The variable can have a value less than but not equal to maxExclusive.</td>
<td>maxExclusive must be greater than or equal to minExclusive. You cannot specify maxInclusive and maxExclusive for the same content type.</td>
</tr>
<tr>
<td>maxInclusive</td>
<td>The upper bound of a range of possible values. The range includes the value you specify. The variable can have a value less than or equal to maxInclusive.</td>
<td>maxInclusive must be greater than or equal to minInclusive. You cannot specify maxInclusive and maxExclusive for the same content type.</td>
</tr>
<tr>
<td>maxLength</td>
<td>The maximum units of length permitted for the variable value.</td>
<td>minLength must be less than or equal to maxLength.</td>
</tr>
</tbody>
</table>
### Constraining Facet

<table>
<thead>
<tr>
<th>Constraining Facet</th>
<th>Description</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>minExclusive</td>
<td>The lower bound of a range of possible values. The range does not include the value you specify. The variable can have a value greater than but not equal to minExclusive.</td>
<td>minExclusive must be less than or equal to maxExclusive. You cannot specify minInclusive and minExclusive for the same content type.</td>
</tr>
<tr>
<td>minInclusive</td>
<td>The lower bound of a range of possible values. The range includes the value you specify. The variable can have a value greater than or equal to minInclusive.</td>
<td>minInclusive must be less than or equal to maxInclusive. You cannot specify minInclusive and minExclusive for the same content type.</td>
</tr>
<tr>
<td>minLength</td>
<td>The minimum units of length permitted for the variable value.</td>
<td>minLength must be less than or equal to maxLength.</td>
</tr>
<tr>
<td>pattern</td>
<td>A pattern (regular expression) that the value of the variable must match. For example, you can use a regular expression to specify that a variable that is a string content constraint match a Social Security number format.</td>
<td></td>
</tr>
<tr>
<td>totalDigits</td>
<td>The maximum number of decimal digits allowed in a value. For example, the totalDigits of the value 999.99 is 5.</td>
<td>totalDigits must be greater than or equal to fractionDigits</td>
</tr>
<tr>
<td>whiteSpace</td>
<td>The white space normalization performed on the variable value. The value of whiteSpace can be one of the following: preserve: No white space normalization is performed. replace: Carriage returns (#xD), line feeds (#xA), and tabs (#x9) are replaced with a single space (#x20). collapse: After the white space normalization specified by replace is performed, sequences of spaces (#x20) and leading and trailing spaces (#x20) are removed.</td>
<td></td>
</tr>
</tbody>
</table>
Note: Previous versions of XML Schema contained the constraining facets `duration`, `encoding`, `period`, `precision`, and `scale`. However, these constraining facets are not included in the recommendation of XML Schema Part 2: Datatypes. The constraining facets `duration`, `encoding`, and `period` were removed. `precision` was renamed `totalDigits`. `scale` was renamed `fractionDigits`. If you view an SAP BC schema created from an XML Schema Definition that used pre-Recommendation version of XML Schema (before May 2001) the Constraints tab and Schema tab will display the constraining facets that were available in the pre-Recommendation version of XML Schema.

Note: On the Constraints tab and Schema tab, the word “fixed” appears next to the name of constraining facets with a fixed value. When a facet has a fixed value, the facet is called a fixed facet. Fixed facets cannot be edited.
Validation Errors and Exceptions

- Overview .......................................................... 628
- Validation Errors ............................................... 628
- Validation Exceptions ........................................... 639
- SAP BC Schema Generation Errors and Warnings .............. 643
APPENDIX H Validation Errors and Exceptions

Overview

This appendix describes error messages that can occur during data validation, SAP BC schema generation, or record generation.

When the validation engine in the SAP BC Server validates objects (XML documents, the pipeline, or records), and the object does not conform to the blueprint or model, the server generates errors and/or exceptions. You might also receive errors and exceptions when creating an SAP BC schema. The following sections describe the errors and exceptions you can receive when performing validation and when creating an SAP BC schema.

Validation Errors

When you perform validation using a built-in service, SAP BC Server returns validation errors in the `errors` output variable if the object is invalid. When you perform input/output validation, SAP BC Server throws an exception if the inputs or outputs are invalid. Error messages are contained in the exception.

Each validation error contains a code and a default message. Error codes that begin with DT are data type validation errors—errors pertaining to the content type constraints applied to the variables. Error codes that begin with NV are node validation errors—errors pertaining to an XML document. Error codes that begin with VV are record validation errors—errors pertaining to the structure of the data values (for example, an invalid record structure).

The following table describes the validation errors you can receive when performing XML, pipeline, or record validation.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT-001</td>
<td>[B2BCORE.0082.9447] Value does not conform to datatype. The value does not match the specified content type.</td>
</tr>
<tr>
<td>DT-002</td>
<td>[B2BCORE.0082.9460] No matching enumeration value. The value is not an item listed in the enumeration field.</td>
</tr>
<tr>
<td>DT-003</td>
<td>[B2BCORE.0082.9463] Length of value is not equal to specified length. The size of the value does not equal the number specified in the length field.</td>
</tr>
<tr>
<td>DT-004</td>
<td>[B2BCORE.0082.9464] Value is shorter than minimum length. The size of the value is less than the number specified in the minLength field.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message and Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>DT-005</td>
<td>[B2BCORE.0082.9465] Value is longer than maximum length. The size of the value is greater than the number specified in the <code>maxLength</code> field.</td>
</tr>
<tr>
<td>DT-006</td>
<td>[B2BCORE.0082.9489] Number of digits is greater than <code>totalDigits</code>. The number of digits in the value is greater than the number specified in the <code>totalDigits</code> field.</td>
</tr>
<tr>
<td>DT-007</td>
<td>[B2BCORE.0082.9490] Number of fraction digits is greater than <code>fractionDigits</code>. The number of digits to the right of the decimal point is greater than the number specified in the <code>fractionDigits</code> field.</td>
</tr>
<tr>
<td>DT-008</td>
<td>[B2BCORE.0082.9491] Value is less than <code>minInclusive</code>. The value is less than the value specified in the <code>minInclusive</code> field.</td>
</tr>
<tr>
<td>DT-009</td>
<td>[B2BCORE.0082.9492] Value is less than or equal to <code>minExclusive</code>. The value is less than or equal to the value specified in the <code>minExclusive</code> field.</td>
</tr>
<tr>
<td>DT-010</td>
<td>[B2BCORE.0082.9493] Value is greater than <code>maxInclusive</code>. The value is greater than the value specified in the <code>maxInclusive</code> field.</td>
</tr>
<tr>
<td>DT-011</td>
<td>[B2BCORE.0082.9494] Value is greater than or equal to <code>maxExclusive</code>. The value is greater than or equal to the value specified in the <code>maxExclusive</code> field.</td>
</tr>
<tr>
<td>DT-012</td>
<td>[B2BCORE.0082.9448] Value does not match pattern. The value does not match the pattern specified in the <code>pattern</code> field.</td>
</tr>
<tr>
<td>DT-013</td>
<td>[B2BCORE.0082.9474] The input has invalid characters. The specified <code>whiteSpace</code> value is invalid. The value of the <code>whiteSpace</code> field can be preserve, replace, or collapse.</td>
</tr>
</tbody>
</table>
### APPENDIX H Validation Errors and Exceptions

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message and Description</th>
</tr>
</thead>
</table>
| DT-Binary001     | [B2BCORE.0082.9293] No matching choice value
The binary value is not an element listed in the choices field. |
| DT-Binary002     | [B2BCORE.0082.9297] Value is shorter than minimum length
Size of the binary value, in octets, is less than the value specified in the minimum length field. |
| DT-Binary003     | [B2BCORE.0082.9298] Value is longer than maximum length
Size of the binary value, in octets, is greater than the value specified in the maximum length field. |
| DT-Binary004     | [B2BCORE.0082.9296] Value is not equal to given length
Size of the binary value, in octets, is not equal to the value specified in the length field. |
| DT-Boolean001    | [B2BCORE.0082.9246] Value does not conform to datatype
The value is not a Boolean. |
| DT-Decimal001    | [B2BCORE.0082.9293] No matching choice value
The decimal value is not an element listed in the choices field. |
| DT-Decimal002    | [B2BCORE.0082.9246] Value does not conform to datatype
The value is not a parsable decimal. |
| DT-Decimal003    | [B2BCORE.0082.9294] Value is less than minimum
The decimal value is less than the value specified in the minimum exclusive or minimum inclusive field. |
| DT-Decimal004    | [B2BCORE.0082.9295] Value is more than maximum
The decimal value is greater than the value specified in the maximum exclusive or maximum inclusive field. |
| DT-Decimal005    | [B2BCORE.0082.9300] Value exceeds totalDigits
The total number of digits in the decimal value is greater than the value specified in the precision field. |
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT-Decimal006</td>
<td>[B2BCORE.0082.9301] Value exceeds scale</td>
</tr>
<tr>
<td></td>
<td>The total number of digits to the right of the decimal point is greater than the value</td>
</tr>
<tr>
<td></td>
<td>specified in the \texttt{scale} field.</td>
</tr>
<tr>
<td>DT-Double001</td>
<td>[B2BCORE.0082.9293] No matching choice value</td>
</tr>
<tr>
<td></td>
<td>The double value is not an element listed in the \texttt{choices} field.</td>
</tr>
<tr>
<td>DT-Double002</td>
<td>[B2BCORE.0082.9294] Value is less than minimum</td>
</tr>
<tr>
<td></td>
<td>The double value is less than the value specified in the \texttt{minimum exclusive} or</td>
</tr>
<tr>
<td></td>
<td>\texttt{minimum inclusive} field.</td>
</tr>
<tr>
<td>DT-Double003</td>
<td>[B2BCORE.0082.9295] Value is more than maximum</td>
</tr>
<tr>
<td></td>
<td>The double value is greater than the value specified in the \texttt{maximum exclusive}</td>
</tr>
<tr>
<td></td>
<td>or \texttt{maximum inclusive} field.</td>
</tr>
<tr>
<td>DT-Double004</td>
<td>[B2BCORE.0082.9246] Value does not conform to datatype</td>
</tr>
<tr>
<td></td>
<td>The value is not a parsable double.</td>
</tr>
<tr>
<td>DT-Float001</td>
<td>[B2BCORE.0082.9293] No matching choice value</td>
</tr>
<tr>
<td></td>
<td>The float value is not an element listed in the \texttt{choices} field.</td>
</tr>
<tr>
<td>DT-Float002</td>
<td>[B2BCORE.0082.9294] Value is less than minimum</td>
</tr>
<tr>
<td></td>
<td>The float value is less than the value specified in the \texttt{minimum exclusive} or</td>
</tr>
<tr>
<td></td>
<td>\texttt{minimum inclusive} field.</td>
</tr>
<tr>
<td>DT-Float003</td>
<td>[B2BCORE.0082.9295] Value is more than maximum</td>
</tr>
<tr>
<td></td>
<td>The float value is greater than the value specified in the \texttt{maximum exclusive}</td>
</tr>
<tr>
<td></td>
<td>or \texttt{maximum inclusive} field.</td>
</tr>
<tr>
<td>DT-Float004</td>
<td>[B2BCORE.0082.9246] Value does not conform to datatype</td>
</tr>
<tr>
<td></td>
<td>The value is not a parsable float.</td>
</tr>
<tr>
<td>DT-Int001</td>
<td>[B2BCORE.0082.9293] No matching choice value</td>
</tr>
<tr>
<td></td>
<td>The int value is not an element listed in the \texttt{choices} field.</td>
</tr>
<tr>
<td>DT-Int002</td>
<td>[B2BCORE.0082.9294] Value is less than minimum</td>
</tr>
<tr>
<td></td>
<td>The int value is less than the value specified in the \texttt{minimum exclusive} or</td>
</tr>
<tr>
<td></td>
<td>\texttt{minimum inclusive} field.</td>
</tr>
</tbody>
</table>
## Validation Errors and Exceptions

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT-Int003</td>
<td>[B2BCORE.0082.9295] Value is more than maximum int value is greater than the value specified in the <strong>maximum exclusive</strong> or <strong>maximum inclusive</strong> field.</td>
</tr>
<tr>
<td>DT-Int004</td>
<td>[B2BCORE.0082.9246] Value does not conform to datatype The value is not a parsable int.</td>
</tr>
<tr>
<td>DT-INTEGER001</td>
<td>[B2BCORE.0082.9293] No matching choice value The integer value is not an element listed in the <strong>choices</strong> field.</td>
</tr>
<tr>
<td>DT-INTEGER002</td>
<td>[B2BCORE.0082.9246] Value does not conform to datatype The value is not a parsable integer.</td>
</tr>
<tr>
<td>DT-INTEGER003</td>
<td>[B2BCORE.0082.9294] Value is less than minimum The integer value is less than the value specified in the <strong>minimum exclusive</strong> or <strong>minimum inclusive</strong> field.</td>
</tr>
<tr>
<td>DT-INTEGER004</td>
<td>[B2BCORE.0082.9295] Value is more than maximum The integer value is greater than the value specified in the <strong>maximum exclusive</strong> or <strong>maximum inclusive</strong> field.</td>
</tr>
<tr>
<td>DT-Long001</td>
<td>[B2BCORE.0082.9293] No matching choice value The long value is not an element listed in the <strong>choices</strong> field.</td>
</tr>
<tr>
<td>DT-Long002</td>
<td>[B2BCORE.0082.9294] Value is less than minimum The long value is less than the value specified in the <strong>minimum exclusive</strong> or <strong>minimum inclusive</strong> field.</td>
</tr>
<tr>
<td>DT-Long003</td>
<td>[B2BCORE.0082.9295] Value is more than maximum The long value is greater than the value specified in the <strong>maximum exclusive</strong> or <strong>maximum inclusive</strong> field.</td>
</tr>
<tr>
<td>DT-Long004</td>
<td>[B2BCORE.0082.9246] Value does not conform to datatype The value is not a parsable long.</td>
</tr>
<tr>
<td>DT-List001</td>
<td>[B2BCORE.0082.9293] No matching choice value The sequence of values in the list is not an element in the <strong>choices</strong> field.</td>
</tr>
</tbody>
</table>
## Validation Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT-List002</td>
<td>[B2BCORE.0082.9297] Value is shorter than minimum length Size of the list is less than the value specified in the <em>minimum length</em> field.</td>
</tr>
<tr>
<td>DT-List003</td>
<td>[B2BCORE.0082.9298] Value is longer than maximum length Size of the list is greater than the value specified in the <em>maximum length</em> field.</td>
</tr>
<tr>
<td>DT-List004</td>
<td>[B2BCORE.0082.9299] Datatype definition is missing Datatype or simple type definition is not found. It is possible that a dependent SAP BC schema that contains this datatype definition was removed from the SAP BC Namespace.</td>
</tr>
<tr>
<td>DT-RecurringDuration001</td>
<td>[B2BCORE.0082.9293] No matching choice value The recurring duration value is not an element listed in the <em>choices</em> field.</td>
</tr>
<tr>
<td>DT-RecurringDuration002</td>
<td>[B2BCORE.0082.9294] Value is less than minimum The recurring duration value is less than the value specified in the <em>minimum exclusive</em> or <em>minimum inclusive</em> field.</td>
</tr>
<tr>
<td>DT-RecurringDuration003</td>
<td>[B2BCORE.0082.9295] Value is more than maximum The recurring duration value is greater than the value specified in the <em>maximum exclusive</em> or <em>maximum inclusive</em> field.</td>
</tr>
<tr>
<td>DT-RecurringDuration004</td>
<td>[B2BCORE.0082.9246] Value does not conform to datatype The recurring duration value does not match the pattern specified in the <em>pattern</em> field.</td>
</tr>
<tr>
<td>DT-STR001</td>
<td>[B2BCORE.0082.9293] No matching choice value The string value is not an element listed in the <em>choices</em> field.</td>
</tr>
<tr>
<td>DT-STR002</td>
<td>[B2BCORE.0082.9297] Value is shorter than minimum length The size of the string, in characters, is less than the value specified in the <em>minimum length</em> field.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message and Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DT-STR003</td>
<td>[B2BCORE.0082.9298] Value is longer than maximum length</td>
</tr>
<tr>
<td></td>
<td>The size of the string, in characters, is greater than the value specified in the maximum length field.</td>
</tr>
<tr>
<td>DT-STR004</td>
<td>[B2BCORE.0082.9307] Does not match pattern(s)</td>
</tr>
<tr>
<td></td>
<td>The string value does not match the pattern specified in the pattern field.</td>
</tr>
<tr>
<td>DT-Time001</td>
<td>[B2BCORE.0082.9293] No matching choice value</td>
</tr>
<tr>
<td></td>
<td>The time value is not an element listed in the choices field.</td>
</tr>
<tr>
<td>DT-Time002</td>
<td>[B2BCORE.0082.9294] Value is less than minimum</td>
</tr>
<tr>
<td></td>
<td>The time value is less than the value specified in the minimum exclusive or minimum inclusive field.</td>
</tr>
<tr>
<td>DT-Time003</td>
<td>[B2BCORE.0082.9295] Value is more than maximum</td>
</tr>
<tr>
<td></td>
<td>The time value is greater than the value specified in the maximum exclusive or maximum inclusive field.</td>
</tr>
<tr>
<td>DT-Time004</td>
<td>[B2BCORE.0082.9246] Value does not conform to datatype</td>
</tr>
<tr>
<td></td>
<td>The time value does not match the pattern specified in the pattern field.</td>
</tr>
<tr>
<td>DT-TimeDuration001</td>
<td>[B2BCORE.0082.9293] No matching choice value</td>
</tr>
<tr>
<td></td>
<td>The time duration value is not an element listed in the choices field.</td>
</tr>
<tr>
<td>DT-TimeDuration002</td>
<td>[B2BCORE.0082.9294] Value is less than minimum</td>
</tr>
<tr>
<td></td>
<td>The time duration value is less than the value specified in the minimum exclusive or minimum inclusive field.</td>
</tr>
<tr>
<td>DT-TimeDuration003</td>
<td>[B2BCORE.0082.9295] Value is more than maximum</td>
</tr>
<tr>
<td></td>
<td>The time duration value is greater than the value specified in the maximum exclusive or maximum inclusive field.</td>
</tr>
<tr>
<td>DT-TimeDuration004</td>
<td>[B2BCORE.0082.9246] Value does not conform to datatype</td>
</tr>
<tr>
<td></td>
<td>The time duration value does not match the pattern specified in the pattern field.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message and Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DT-TimePeriod001</td>
<td>[B2BCORE.0082.9246] Values does not conform to data type</td>
</tr>
<tr>
<td></td>
<td>The time period value does not match data type specified in the SAP BC schema.</td>
</tr>
<tr>
<td>DT-TimePeriod002</td>
<td>[B2BCORE.0082.9293] No matching choice value</td>
</tr>
<tr>
<td></td>
<td>The time period value is not an element listed in the choices field.</td>
</tr>
<tr>
<td>DT-TimePeriod003</td>
<td>[B2BCORE.0082.9294] Value is less than minimum</td>
</tr>
<tr>
<td></td>
<td>The time duration value is less than the value specified in the minimum exclusive or</td>
</tr>
<tr>
<td></td>
<td>minimum inclusive field.</td>
</tr>
<tr>
<td>DT-TimePeriod004</td>
<td>[B2BCORE.0082.9295] Value is more than maximum</td>
</tr>
<tr>
<td></td>
<td>The time duration value is greater than the value specified in the maximum exclusive or</td>
</tr>
<tr>
<td></td>
<td>maximum inclusive field.</td>
</tr>
<tr>
<td></td>
<td>Document cannot be parsed. It is possible that the XML document being validated is not</td>
</tr>
<tr>
<td></td>
<td>well formed.</td>
</tr>
<tr>
<td>NV-002</td>
<td>[B2BCORE.0082.9002] Unable to retrieve root element</td>
</tr>
<tr>
<td></td>
<td>XML document is empty.</td>
</tr>
<tr>
<td>NV-003</td>
<td>[B2BCORE.0082.9003] Unable to locate a matching element declaration</td>
</tr>
<tr>
<td></td>
<td>An undeclared element node is found in the XML document.</td>
</tr>
<tr>
<td>NV-004</td>
<td>[B2BCORE.0082.9004] [attributes] property must be empty</td>
</tr>
<tr>
<td></td>
<td>This element node contains attributes that are not expected.</td>
</tr>
<tr>
<td>NV-005</td>
<td>[B2BCORE.0082.9005] Element information items are not allowed in [children] property</td>
</tr>
<tr>
<td></td>
<td>This element node contains elements that are not expected.</td>
</tr>
<tr>
<td>NV-006</td>
<td>[B2BCORE.0082.9006] Unable to locate a matching attribute declaration</td>
</tr>
<tr>
<td></td>
<td>This element node contains an undeclared attribute.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message and Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>NV-007</td>
<td>[B2BCORE.0082.9007] Missing Attribute Information Item</td>
</tr>
<tr>
<td></td>
<td>A required attribute is not found in this element node.</td>
</tr>
<tr>
<td>NV-008</td>
<td>[B2BCORE.0082.9008] Invalid chunk of CII (s) - does not match #FIXED value</td>
</tr>
<tr>
<td></td>
<td>The XML document contains an invalid element body or attribute value. Specifically, the element body or attribute value does not match a fixed value found in the declaration. (CII = Character Information Item)</td>
</tr>
<tr>
<td>NV-009</td>
<td>[B2BCORE.0082.9009] Child element <code>elementName</code> at position <code>location</code> is unexpected.</td>
</tr>
<tr>
<td></td>
<td>Element <code>elementName</code> is not a valid child or the sequence of child nodes does not satisfy the grammar specified in the definition.</td>
</tr>
<tr>
<td></td>
<td>[B2BCORE.0082.9010] Incomplete content–one or more child elements are expected.</td>
</tr>
<tr>
<td></td>
<td>Element <code>elementName</code> is not a valid child or the sequence of child nodes does not satisfy the grammar specified in the definition.</td>
</tr>
<tr>
<td>NV-010</td>
<td>[B2BCORE.0082.9011] Unable to locate attribute declaration</td>
</tr>
<tr>
<td></td>
<td>An attribute declaration is not found. It is possible that a dependent SAP BC schema that contains this attribute declaration was removed from the SAP BC Namespace.</td>
</tr>
<tr>
<td></td>
<td>For example, SAP BC schema <code>pub.schema.w3c:datatypes</code> references an attribute declaration (say <code>xml:lang</code>) in <code>pub.schema.w3c:xml</code>. If you receive this error, it is possible that <code>pub.schema.w3c:xml</code> was removed.</td>
</tr>
<tr>
<td>NV-011</td>
<td>[B2BCORE.0082.9012] Unable to locate type definition</td>
</tr>
<tr>
<td></td>
<td>A simple or complex type definition is not found. It is possible that a dependent SAP BC schema that contains this type definition was removed from the SAP BC Namespace.</td>
</tr>
<tr>
<td></td>
<td>For example, SAP BC schema <code>pub.schema.w3c:structures</code> references a type definition in <code>pub.schema.w3c:datatypes</code>. If you receive this error, it is possible that <code>pub.schema.w3c:datatypes</code> was removed.</td>
</tr>
</tbody>
</table>
### Validation Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NV-012</td>
<td>[B2BCORE.0082.9014] Unable to locate element declaration</td>
</tr>
<tr>
<td></td>
<td>An element declaration is not found. It is possible that a dependent SAP BC schema that contains the element declaration was removed from the SAP BC Namespace.</td>
</tr>
<tr>
<td>NV-013</td>
<td>[B2BCORE.0082.9016] Unable to resolve QName: [URIa] name</td>
</tr>
<tr>
<td></td>
<td>The schema processor uses the namespaces declared in the instance document to resolve a QName to:</td>
</tr>
<tr>
<td></td>
<td>{Namespace URI} Local Name</td>
</tr>
<tr>
<td></td>
<td>However, the schema processor is unable to resolve the QName using the namespace declarations in the instance document.</td>
</tr>
<tr>
<td>NV-014</td>
<td>[B2BCORE.0082.9017] %Type-b% is not validly derived from %Type-a%</td>
</tr>
<tr>
<td></td>
<td>This error occurs when %Type-b% is used in a context where %Type-a% is expected, and one of the following is true:</td>
</tr>
<tr>
<td></td>
<td>%Type-b% is not the same as %Type-a%</td>
</tr>
<tr>
<td></td>
<td>-OR-</td>
</tr>
<tr>
<td></td>
<td>%Type-b% is not validly derived from %Type-a%</td>
</tr>
<tr>
<td>NV-015</td>
<td>[B2BCORE.0082.9018] %Type-i% is an abstract type and cannot be used directly to validate content</td>
</tr>
<tr>
<td></td>
<td>%Type-i% is an abstract type that has been either declared or nominated. Abstract types cannot be used to directly validate the contents of an element.</td>
</tr>
<tr>
<td>NV-016</td>
<td>[B2BCORE.0082.9019] %Element Decl A% is an abstract element and cannot appear in an instance</td>
</tr>
<tr>
<td></td>
<td>The element declaration is an abstract element and cannot appear in an instance document.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message and Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>NV-017</td>
<td>[B2BCORE.0082.9020] QName - xsi:type is used incorrectly (declared type is anonymous) xsi:type is used incorrectly. A type is nominated using xsi:type for an element whose declaration contains an anonymous type. A new type cannot be derived from an anonymous type because an anonymous type is not a named type.</td>
</tr>
<tr>
<td>NV-018</td>
<td>[B2BCORE.0082.9021] Contains invalid text The schema processor encountered an invalid piece of text. It is possible that the instance document contains a simple type where element declarations are interspersed with text. Simple types cannot contain element declarations.</td>
</tr>
<tr>
<td>VV-001</td>
<td>[B2BCORE.0082.9025] Missing Object A required record or variable is missing from the input</td>
</tr>
<tr>
<td>VV-002</td>
<td>[B2BCORE.0082.9026] Undefined Object found A record contains an orphan variable. (This message only appears if you cleared the Allow unspecified fields check box on the Constraints tab of the Variable Properties dialog box.)</td>
</tr>
<tr>
<td>VV-003</td>
<td>[B2BCORE.0082.9027] Dimension mismatch, List expected The value being validated is a scalar value or a multi-dimensional array (String Table), however the variable it is being validated against is a list (one-dimensional array). [B2BCORE.0082.9028] Dimension mismatch, Single item expected. The value being validated is an array value (a list or a table). The variable it is being validated against is scalar. [B2BCORE.0082.9029] Dimension mismatch, Table expected The value being validated is a scalar value or a one-dimensional array (list). The variable it is being validated against is a two-dimensional array (table).</td>
</tr>
<tr>
<td>VV-004</td>
<td>[B2BCORE.0082.9030] Type mismatch, String expected The value is being validated against a String variable, but the SAP BC data type of the value is not a String.</td>
</tr>
</tbody>
</table>
Validation Exceptions

At run time, the service performing validation either succeeds or fails. If the service fails, SAP BC Server throws a validation exception. A validation exception is generated if one of the following is true:

- Errors are detected in the object (XML document, pipeline, or Record) that is passed (e.g., null value).
- The basic validation contract is violated (e.g., a binary tree is passed instead of a record as expected).
- You specify that the service should fail if the object to be validated (XML document, pipeline, or record) did not conform to the SAP BC schema or record (e.g., failIfInvalid = true). If this is the reason for the exception, SAP BC Server inserts the validation errors into the exception message.

The following table identifies and describes the validation exceptions that can be generated.

<table>
<thead>
<tr>
<th>Default Exception Message</th>
<th>When is it thrown?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[B2BSERV.0062.9021] object is null</td>
<td>Real-time and design time</td>
<td>The object to be validated does not exist in the pipeline.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Solution:</strong> Map a node or record variable to the object variable in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Service In.</td>
</tr>
<tr>
<td>[B2BSERV.0062.9022] %NSName% - object does not exists</td>
<td>Real-time and design time</td>
<td>The record or SAP BC schema specified for the conformsTo variable does not exist in the SAP BC Namespace.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Solution:</strong> Change the value of conformsTo to be a record or SAP BC schema.</td>
</tr>
</tbody>
</table>
## Validation Errors and Exceptions

<table>
<thead>
<tr>
<th>Default Exception Message</th>
<th>When is it thrown?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[B2BSERV.0062.9024] SAP BC Server does not support this type of validation (may or may not support in the future)</td>
<td>Real-time and design time</td>
<td>The object to be validated is not one of the types supported by validation. <strong>Solution:</strong> Currently, only XML, pipeline, and record validation is supported.</td>
</tr>
<tr>
<td>[B2BSERV.0062.9202] Invalid minimum length</td>
<td>Design-time</td>
<td>The min length is either not a parsable number or conflicts with max length. <strong>Solution:</strong> Change the min length value and make sure that it is within the allowed range for the content type.</td>
</tr>
<tr>
<td>[B2BSERV.0062.9203] Invalid maximum length</td>
<td>Design-time</td>
<td>The max length is either not a parsable number or conflicts with minimum length. <strong>Solution:</strong> Change the max length value and make sure that it is within the allowed range for the content type.</td>
</tr>
<tr>
<td>[B2BSERV.0062.9211] Invalid Regex</td>
<td>Design-time</td>
<td>The pattern is an invalid Perl regular expression. <strong>Solution:</strong> Modify the pattern and make sure that it is Perl regular expression.</td>
</tr>
<tr>
<td>[B2BSERV.0062.9212] Invalid: minInclusive</td>
<td>Design-time</td>
<td>The specified minInclusive value is invalid. For example, the minInclusive value for a variable with content type constraint of short must also be a short. <strong>Solution:</strong> Make sure the minInclusive is a valid number for the specified content type.</td>
</tr>
<tr>
<td>[B2BSERV.0062.9213] Invalid: minExclusive</td>
<td>Design-time</td>
<td>The specified minExclusive value is invalid. For example, the minExclusive value for a variable with content type constraint of short must also be a short. <strong>Solution:</strong> Make sure the minExclusive is a valid number for the specified content type.</td>
</tr>
<tr>
<td>Default Exception Message</td>
<td>When is it thrown?</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>[B2BSERV.0062.9214] Invalid: maxInclusive</td>
<td>Design-time</td>
<td>The specified maxInclusive value is invalid. For example, the maxInclusive value for a variable with content type constraint of short must also be a short.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Solution:</strong> Make sure the maxInclusive is a valid number for the specified content type.</td>
</tr>
<tr>
<td>[B2BSERV.0062.9215] Invalid: maxExclusive</td>
<td>Design-time</td>
<td>The specified maxExclusive value is invalid. For example, the maxExclusive value for a variable with content type constraint of short must also be a short.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Solution:</strong> Make sure the maxExclusive is a valid number for the specified content type.</td>
</tr>
<tr>
<td>[B2BSERV.0062.9230] Invalid scale</td>
<td>Design-time</td>
<td>The value of a constraining facet has an invalid scale.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Solution:</strong> Examine the constraining facet values to make sure the values do not conflict.</td>
</tr>
<tr>
<td>[B2BSERV.0062.9231] Exceeds precision: minInclusive</td>
<td>Design-time</td>
<td>The value of the minInclusive constraining facet exceeds the value specified for totalDigits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Solution:</strong> Change the value of minInclusive or totalDigits to make sure the values do not conflict.</td>
</tr>
<tr>
<td>[B2BSERV.0062.9232] Exceeds precision: minExclusive</td>
<td>Design-time</td>
<td>The value of the minExclusive constraining facet exceeds the value specified for totalDigits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Solution:</strong> Change the value of minExclusive or totalDigits to make sure the values do not conflict.</td>
</tr>
<tr>
<td>[B2BSERV.0062.9233] Exceeds precision: maxInclusive</td>
<td>Design-time</td>
<td>The value of the maxInclusive constraining facet exceeds the value specified for totalDigits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Solution:</strong> Change the value of maxInclusive or totalDigits to make sure the values do not conflict.</td>
</tr>
<tr>
<td>Default Exception Message</td>
<td>When is it thrown?</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>[B2BSERV.0062.9234]</td>
<td>Design-time</td>
<td>The value of the maxExclusive constraining facet exceeds the value specified for totalDigits.</td>
</tr>
<tr>
<td>Exceeds precision: maxExclusive</td>
<td></td>
<td><strong>Solution:</strong> Change the value of maxExclusive or totalDigits to make sure the values do not conflict.</td>
</tr>
<tr>
<td>[B2BSERV.0062.9235]</td>
<td>Design-time</td>
<td>A content type constraint is defined in terms of the collective set of constraining facet values. Together, these values determine the allowed values and properties of the content type. The constraining facet value you just specified may conflict with other constraining facet values. For example, if you specify length for the string content type, you cannot specify min length or max length.</td>
</tr>
<tr>
<td>Invalid choice item</td>
<td></td>
<td><strong>Solution:</strong> Examine the constraining facet values to make sure the values do not conflict with each other.</td>
</tr>
<tr>
<td>[B2BSERV.0062.9302]</td>
<td>Design-time</td>
<td>Invalid condition. The maximum value is less than the minimum value. (maximum &lt; minimum)</td>
</tr>
<tr>
<td>Maximum is less than minimum</td>
<td></td>
<td><strong>Solution:</strong> Change the maximum or minimum value to eliminate the conflict.</td>
</tr>
<tr>
<td>[B2BSERV.0062.9303]</td>
<td>Design-time</td>
<td>The minimum value is out of the valid range for the content type.</td>
</tr>
<tr>
<td>Minimum is out of range</td>
<td></td>
<td><strong>Solution:</strong> Change the minimum value (min exclusive, min inclusive, or min length) and make sure that it is within the allowed range for the content type.</td>
</tr>
<tr>
<td>[B2BSERV.0062.9304]</td>
<td>Design-time</td>
<td>The specified maximum value is out of the valid range for the content type.</td>
</tr>
<tr>
<td>Maximum is out of range</td>
<td></td>
<td><strong>Solution:</strong> Change the maximum value (max exclusive, max inclusive or max length) and make sure that it is within the allowed range for the content type.</td>
</tr>
</tbody>
</table>
SAP BC Schema Generation Errors and Warnings

When you create an SAP BC schema from a DTD or an XML Schema definition, you may receive errors or warnings. The following table identifies and describes the errors you might receive when creating an SAP BC schema. Error or warning codes that begin with DTDC are errors that occur when you generate an SAP BC schema from a DTD (or from an XML document that references an existing DTD). Error or warning codes that begin with XSDC are errors that occur when you generate an SAP BC schema from an XML Schema definition.

**Note:** You might also receive these errors and warnings when you generate a record or flow service from an XML Schema definition, DTD, or XML document that references a DTD. For more information about creating a record, see “Creating a Record” on page 200. For more information about creating a flow service, see “Creating a New Flow Service” on page 76.

<table>
<thead>
<tr>
<th>Code</th>
<th>Message and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTDC-001</td>
<td>[B2BCORE.0082.9501] DTD is empty&lt;br&gt;An error. DTD is empty.</td>
</tr>
<tr>
<td>DTDC-002</td>
<td>[B2BCORE.0082.9502] NS Declaration is missing for prefix ‘prefix’&lt;br&gt;A warning. An XML Namespace prefix is used without declaring it.</td>
</tr>
<tr>
<td>DTDC-003</td>
<td>[B2BCORE.0082.9503] Parser says 'message'&lt;br&gt;A warning. DTD cannot be parsed. It is possible the DTD does not satisfy XML 1.0.</td>
</tr>
<tr>
<td>DTDC-004</td>
<td>[B2BCORE.0082.9504] Declaration not found&lt;br&gt;A warning. An element type is used without being declared.</td>
</tr>
<tr>
<td>DTDC-005</td>
<td>[B2BCORE.0082.9505] Name collision&lt;br&gt;A warning. An element type declaration is found in another schema with the same XML Namespace.</td>
</tr>
<tr>
<td>XSDC-001</td>
<td>[B2BCORE.0082.9703] Duplicate declaration found in this schema definition&lt;br&gt;An error. A duplicate attribute or element declaration is found in the XML Schema definition.</td>
</tr>
<tr>
<td>XSDC-002</td>
<td>[B2BCORE.0082.9704] Duplicate definition found in this schema definition&lt;br&gt;An error. A duplicate simple type or complex type definition is found in the XML Schema definition.</td>
</tr>
<tr>
<td>Code</td>
<td>Message and Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XSDC-003</td>
<td>[B2BCORE.0082.9705] Definition not found</td>
</tr>
<tr>
<td></td>
<td>An error. A simple type or complex type definition is missing from the XML Schema definition.</td>
</tr>
<tr>
<td>XSDC-004</td>
<td>[B2BCORE.0082.9706] Declaration not found</td>
</tr>
<tr>
<td></td>
<td>An error. An element or attribute declaration is missing from the XML Schema definition.</td>
</tr>
<tr>
<td>XSDC-005</td>
<td>[B2BCORE.0082.9707] Base type definition not found</td>
</tr>
<tr>
<td></td>
<td>An error. A base type definition that is used to derive either a simple type or complex type is missing from the XML Schema definition.</td>
</tr>
<tr>
<td>XSDC-006</td>
<td>[B2BCORE.0082.9708] Type derivation not OK</td>
</tr>
<tr>
<td></td>
<td>An error. As per the spec(s) from the W3C, the “type derivation is not OK.”</td>
</tr>
<tr>
<td></td>
<td>[B2BCORE.0082.9708] attributeName: attribute declaration to be restricted is not found in the base type definition</td>
</tr>
<tr>
<td></td>
<td>In the type derivation, an attribute is restricted, however, this attribute declaration is not found in the base type definition.</td>
</tr>
<tr>
<td></td>
<td>[B2BCORE.0082.9708] attributeName: attribute declaration to be prohibited is not found in the base type definition</td>
</tr>
<tr>
<td></td>
<td>In a type derivation, an attribute is prohibited; however, this attribute declaration is not found in the base type definition.</td>
</tr>
<tr>
<td>XSDC-007</td>
<td>[B2BCORE.0082.9711] Built-in type not found for datatype</td>
</tr>
<tr>
<td></td>
<td>The datatype is not one of the content types built into SAP BC Server.</td>
</tr>
<tr>
<td>XSDC-008</td>
<td>[B2BCORE.0082.9712] Incorrect facet (s) specified</td>
</tr>
<tr>
<td></td>
<td>Facets applied to the content type are incorrect or cannot be used with the content type applied to the variable.</td>
</tr>
<tr>
<td>XSDC-009</td>
<td>[B2BCORE.0082.9713] Unable to resolve QName</td>
</tr>
<tr>
<td></td>
<td>Incorrect QName.</td>
</tr>
</tbody>
</table>
**SAP BC Schema Generation Errors and Warnings**

<table>
<thead>
<tr>
<th>Code</th>
<th>Message and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSDC-080</td>
<td>[B2BCORE.0082.9701] Duplicate declaration found in another schema with the same target namespace. A warning. A duplicate attribute or element declaration is found in another schema with the same target namespace.</td>
</tr>
<tr>
<td>XSDC-081</td>
<td>[B2BCORE.0082.9702] Duplicate definition found in another schema with the same target namespace. A warning. A duplicate simple type or complex type definition is found in another schema with the same target namespace.</td>
</tr>
</tbody>
</table>
WSDL Errors and Warnings

- Overview ................................................................. 648
- WSDL Related Errors and Warnings .......................... 649
Overview

When you use the New command to generate a Web Service Connector from a WSDL document, SAP BC Developer displays a message stating whether the Web Service Connector generated successfully.

If SAP BC Developer generated the Web Service Connector successfully, but warnings occurred, SAP BC Developer displays the message

“WSDL Connector created successfully but with warnings.”

If SAP BC Developer could not create the Web Service Connector or could not create some flow steps in the Web Service Connector, SAP BC Developer displays the message

“Error occurred while creating Web Service Connector.”

When you click the Details button, SAP BC Developer lists the name and location of the WSDL for which you are generating a Web Service Connector, the path name to the WSDL element for which the error or warning occurred, and the error or warning code.

**SAP BC Developer Message**

**URL:** D:\WSDL\WSDL_Stockquote.wsdl

**Path name:**

- Top Level Document: D:\WSDL\WSDL_Stockquote.wsdl
- WSDL code: SERV-9034
  - [BIZSERV.0892.9034] Warning: Found port with an invalid binding reference, the port was not generated.

**URL:** D:\WSDL\WSDL_Stockquote.wsdl

**Path name:**

- PortType: StockQuotePortType
  - Operation: GetLastTradePrice
  - WSDL code: SERV-9013
  - [BIZSERV.0892.9013] Warning: The operation's binding does not have any ports, no ports were generated for the Web Service Connector.

**Note:** SAP BC Developer might generate some of the flow steps in the Web Service Connector or some of the supporting SAP BC elements (records, folders, or SAP BC schemas) before it encounters errors or warnings. The generated elements appear in the Service Browser.
WSDL Related Errors and Warnings

Following are the error messages and warning messages that can occur when SAP BC Developer creates a Web Service Connector or when SAP BC Developer generates a WSDL document.

[B2BCORE.0092.9001] Server Error: {0}

A server error occurred while Developer was generating the Web Service Connector. Click the Details button to view the errors.

[B2BCORE.0092.9002] Warning: Document does not contain service element, no ports were generated for any Web Service Connector.

The WSDL document does not contain a <service> element, and therefore does not contain any <port> elements. (A <service> element is a collection of <port> elements.)

SAP BC Developer generates the Web Service Connector, but the Web Service Connector will not specify any flow MAP steps for setting the port information. In the Web Service Connector the BRANCH on '/_port' step contains a child portName MAP step for each unique <port> associated with the <operation>. The BRANCH on '/_port' step will not contain child portName MAP steps. The Web Service Connector cannot execute successfully without port information because the port information specifies the network address for invoking the Web Service.

[B2BCORE.0092.9003] Error: SOAP binding does not contain extended element http://schemas.xmlsoap.org/wsdl/soap/binding, binding was not created.

The <binding> element is missing the <soap:binding> element. If the WSDL document specifies SOAP as a protocol, the <binding> element must contain <soap:binding> as the first child element.

SAP BC Developer generates the Web Service Connector, but the Web Service Connector does not contain a SEQUENCE step that corresponds to this binding. (In the Web Service Connector, the BRANCH on '/binding' step contains a child SEQUENCE step for each unique <binding> associated with an <operation>.)

[B2BCORE.0092.9004] Error: SOAP binding does not contain transport value, binding was not created.

In the WSDL document, the <soap:binding> element does not specify a value for the transport attribute. The transport attribute exists, but no value is specified. The transport value indicates which transport of SOAP the binding uses. It is required for a SOAP binding.

SAP BC Developer generates the Web Service Connector, but the Web Service Connector does not contain a SEQUENCE step that corresponds to this binding. In the Web Service Connector, the BRANCH on '/binding' step contains a child SEQUENCE step for each unique <binding> associated with an <operation>.
Appendix I  WSDL Errors and Warnings

[B2BCORE.0092.9005] Error: SOAP binding has an unsupported transport value, binding was not created.

In the WSDL document, the transport attribute in <soap:binding> element specifies an unsupported SOAP transport. SAP BC Developer can generate a binding for a SOAP binding only if the transport value is http://schemas.xmlsoap.org/soap/http.

SAP BC Developer generates the Web Service Connector, but the Web Service Connector does not contain a SEQUENCE step that corresponds to this binding. In the Web Service Connector, the BRANCH on '/binding' step contains a child SEQUENCE step for each unique <binding> associated with an <operation>.

[B2BCORE.0092.9006] Error: SOAP binding does not contain a transport attribute, binding was not created.

In the WSDL document, the <soap:binding> element does not contain a transport attribute. The transport value indicates which transport of SOAP the binding uses. It is required for a SOAP binding. The transport attribute value must be http://schemas.xmlsoap.org/soap/http.

SAP BC Developer generates the Web Service Connector, but the Web Service Connector does not contain a SEQUENCE step that corresponds to this binding. In the Web Service Connector, the BRANCH on '/binding' step contains a child SEQUENCE step for each unique <binding> associated with an <operation>.

[B2BCORE.0092.9007] Error: SOAP binding has an unrecognized style value, binding was not created.

In the WSDL document, the <soap:binding> element specifies a value other than rpc or document for the style attribute.

SAP BC Developer generates the Web Service Connector, but the Web Service Connector does not contain a SEQUENCE step that corresponds to this binding. In the Web Service Connector, the BRANCH on '/binding' step contains a child SEQUENCE step for each unique <binding> associated with an <operation>.

[B2BCORE.0092.9008] Error: HTTP binding does not contain extended element http://schemas.xmlsoap.org/wsdl/http(binding), binding was not created.

The <binding> element is missing the <http:binding> element. If the WSDL document specifies HTTP as a protocol, the <binding> element must contain <http:binding> as the first child element.

SAP BC Developer generates the Web Service Connector, but the Web Service Connector does not contain a SEQUENCE step that corresponds to this binding. In the Web Service Connector, the BRANCH on '/binding' step contains a child SEQUENCE step for each unique <binding> associated with an <operation>.
[B2BCORE.0092.9009] Error: HTTP binding does not contain required verb attribute, binding was not created.

In the WSDL document, the `<http:binding>` element does not contain the `verb` attribute. The value of the `verb` attribute must be `GET` or `POST`.

SAP BC Developer generates the Web Service Connector, but the Web Service Connector does not contain a `SEQUENCE` step that corresponds to this binding. In the Web Service Connector, the `BRANCH` on `/binding` step contains a child `SEQUENCE` step for each unique `<binding>` associated with an `<operation>`.

[B2BCORE.0092.9010] Error: HTTP binding has an unsupported verb attribute, binding was not created.

In the WSDL document, the `<http:binding>` element specifies a `verb` attribute value other than `GET` or `POST`.

SAP BC Developer generates the Web Service Connector, but the Web Service Connector does not contain a `SEQUENCE` step that corresponds to this binding. (In the Web Service Connector, the `BRANCH` on `/binding` step contains a child `SEQUENCE` step for each unique `<binding>` associated with an `<operation>`.)

[B2BCORE.0092.9011] Error: Mime binding style is unsupported, binding was not created.

The WSDL document specifies a MIME binding style for the entire `<binding>`. SAP BC Developer only supports the MIME binding style to describe the inputs and outputs of an HTTP binding. SAP BC Developer cannot generate a binding when the MIME binding style is specified outside of the HTTP binding context.

SAP BC Developer generates the Web Service Connector, but the Web Service Connector does not contain a `SEQUENCE` step that corresponds to this binding. (In the Web Service Connector, the `BRANCH` on `/binding` step contains a child `SEQUENCE` step for each unique binding associated with an operation.)

[B2BCORE.0092.9013] Warning: The operation's binding does not have any ports, no ports were generated for the Web Service Connector.

SAP BC Developer cannot find a `<port>` element that corresponds to a particular `<binding>` element. A `<port>` element specifies a network address or endpoint for a binding.

SAP BC Developer generates the Web Service Connector, but does not generate any MAP steps for setting the binding and address information. In a Web Service Connector, the `BRANCH` on `/_port` step contains a child `portName` MAP step for each `<port>` associated with the `<operation>`. When this warning occurs, the `BRANCH` on `/_port` step contains no child `portName` MAP steps.
Appendix I  WSDL Errors and Warnings

[B2BCORE.0092.9014] Warning: The operation does not have any valid ports, no ports were generated for the Web Service Connector.

The WSDL document does not contain any valid `<port>` elements for an `<operation>`. The WSDL document might not contain any `<port>` elements or a `<port>` element might reference a non-existent `<binding>` element. (A `<binding>` element associates a protocol with an `<operation>`.)

SAP BC Developer generates the Web Service Connector, but does not generate any `portName` MAP steps for setting the binding and address information. In a Web Service Connector, the BRANCH on `/_port` step contains a child `portName` MAP step for each `<port>` associated with the `<operation>`. When this warning occurs, the BRANCH on `/_port` step contains no child `portName` MAP steps.

[B2BCORE.0092.9015] Warning: Port does not have a valid binding type, port was not generated.

The WSDL document contains a `<port>` element that does not contain the `binding` attribute.

SAP BC Developer generates the Web Service Connector, but does not generate a MAP step for this port. In a Web Service Connector, the BRANCH on `/_port` step contains a child `portName` MAP step for each `<port>` associated with the `<operation>`. The `portName` MAP step sets the binding and address information for a port.

[B2BCORE.0092.9016] Warning: Port does not have a location value, port was not generated.

Within the `<port>` element, the `<address>` element does not specify a value for the location attribute. The location attribute specifies the network address or endpoint for the service.

SAP BC Developer generates a Web Service Connector, but without the network address, SAP BC Developer cannot generate a MAP step for this port. In a Web Service Connector, the BRANCH on `/_port` step contains a child `portName` MAP step for each `<port>` associated with the `<operation>`. The `portName` MAP step sets the binding and address information for a port.

Note: This warning is the same as [B2BCORE.0092.9019].

[B2BCORE.0092.9017] Warning: Port does not have required address element, port was not generated.

The selected WSDL document does not contain an `<address>` element within the specified `<port>` element. The `<address>` element carries an attribute that specifies the location or network address for of the Web Service.

SAP BC Developer generates a Web Service Connector, but does not generate a MAP step for this port. The `portName` MAP step sets the binding and address information for a port. Without the `<address>` element, SAP BC Developer cannot set the address information and therefore cannot generate a MAP step.
[B2BCORE.0092.9018] Warning: Port does not have required location attribute, port was not generated.

Within the <port> element, the <address> element does not carry the location attribute. The location attribute specifies the network address for the service.

SAP BC Developer generates a Web Service Connector, but does not generate a MAP step for this port. The portName MAP step sets the binding and address information for a port. Without the location attribute, SAP BC Developer cannot set the address information, and therefore cannot generate a MAP step.

[B2BCORE.0092.9019] Warning: Port does not have a location value, port was not generated.

Within the <port> element, the <address> element does not specify a value for the location attribute. The location attribute specifies the network address or endpoint for the service.

SAP BC Developer generates a Web Service Connector, but without the network address, SAP BC Developer cannot generate a MAP step for this port. The portName MAP step sets the binding and address information for a port. In a Web Service Connector, the BRANCH on '/_port' step contains a child portName MAP step for each <port> associated with the <operation>.

Note: This warning is the same as [B2BCORE.0092.9016].

[B2BCORE.0092.9020] Error: Operation is not referenced by any binding, Web Service Connector was not created.

The WSDL document does not specify a <binding> element for the <operation>. Each <operation> within a <portType> element needs to correspond to an <operation> element within a <binding> element. Without a binding, the WSDL document does not provide any information about how to invoke the Web Service.

SAP BC Developer does not generate a Web Service Connector for the <operation>.

[B2BCORE.0092.9021] Error: Input and Output messages missing, invalid operation, Web Service Connector was not created.

In the WSDL document, the <operation> element within a <portType> element does not declare an input message or an output message. An input message is declared using the <input> element. An output message is declared using the <output> element. For SAP BC Developer to generate a Web Service Connector for the <operation>, the <operation> element must identify an input message—the <operation> element must contain an <input> element.

SAP BC Developer does not generate a Web Service Connector for the <operation>.
Appendix I  WSDL Errors and Warnings

[B2BCORE.0092.9022] Error: Input message missing, Notification operations not supported. Web Service Connector was not created.

In the WSDL document, the <operation> element does not declare an input message, but it does declare an output message. In other words, the <operation> element does not contain a child <input> element, but does contain a child <output> element. This structure corresponds to the grammar for a notification operation. SAP BC Developer does not generate Web Service Connectors for notification operations.

[B2BCORE.0092.9023] Error: Output message precedes Input message, Solicit Response operations not supported. Web Service Connector was not created.

In the WSDL document, the <operation> element declares an <output> element (output message) before the <input> element (input message). This describes a solicit-response operation. SAP BC Developer does not generate Web Service Connectors for solicit-response operations.

[B2BCORE.0092.9024] Error: HTTP binding has mime multipart Input. Multipart Input is not supported. Binding was not generated.

The <binding> element specifies a multi-part MIME binding for the operation. SAP BC Developer does not support this type of binding.

SAP BC Developer generates the Web Service Connector, but the Web Service Connector does not contain a SEQUENCE step that corresponds to this binding. In the Web Service Connector, the BRANCH on '/binding' step contains a child SEQUENCE step for each unique binding associated with an operation.

[B2BCORE.0092.9025] Error: HTTP Binding input is of type http:urIReplacement. http:urIReplacement is not supported. Binding was not generated.

The <binding> element specifies <http:urIReplacement> as the binding for the operation input. SAP BC Developer does not support this type of binding for the input message.

SAP BC Developer generates the Web Service Connector, but the Web Service Connector does not contain a SEQUENCE step that corresponds to this binding. In the Web Service Connector, the BRANCH on '/binding' step contains a child SEQUENCE step for each unique binding associated with an operation.

[B2BCORE.0092.9026] Error: HTTP Binding input is of an unknown type. Binding was not generated.

The input binding specifies an unknown binding type. When the protocol is HTTP POST, the <mime:content> element for the input binding must specify text/xml, text/plain, or application/x-www-form-urlencoded for the type attribute. (The <mime:mimeXml> element is also valid for the input binding.) When the protocol is HTTP GET, the input binding must contain the child element <http:urIEncod ed>.

SAP BC Developer generates the Web Service Connector, but the Web Service Connector does not contain a SEQUENCE step that corresponds to this binding. (In the Web Service Connector, the BRANCH on '/binding' step contains a child SEQUENCE step for each unique binding associated with an operation.)
WSDL Related Errors and Warnings

[B2BCORE.0092.9027] Error: HTTP Binding output mime parts are missing. Binding was not generated completely.

The `<binding>` element specifies MIME binding for the output message, but the output binding does not specify a message part for the `<mime:content>` element or the output binding is missing `<mime:part>` elements.

SAP BC Developer generates the Web Service Connector, and generates a SEQUENCE step that corresponds to this binding. (In the Web Service Connector, the BRANCH on `/binding` step contains a child SEQUENCE step for each unique binding associated with an operation.) However, SAP BC Developer does not generate a complete binding because the output binding in the WSDL document does not provide the part name information that SAP BC Developer needs to map out the service results to variables in the pipeline. Specifically, the Web Service Connector does not contain the BRANCH on `/numParts` step for this binding.

[B2BCORE.0092.9028] Error: HTTP Binding output mime part is missing its type. Binding was not generated completely.

The `<binding>` element specifies MIME binding for the output message, but the `<mime:content>` element for the output binding does not specify a value for the `type` attribute. The `type` attribute specifies the MIME type.

SAP BC Developer generates the Web Service Connector, and generates a SEQUENCE step that corresponds to this binding. (In the Web Service Connector, the BRANCH on `/binding` step contains a child SEQUENCE step for each unique binding associated with an operation.) However, SAP BC Developer does not generate a complete binding because the output binding in the WSDL document does not provide the part name information that SAP BC Developer needs to map out the service results to variables in the pipeline. Specifically, in the Web Service Connector, the BRANCH on `/loopCount` step does not have a child SEQUENCE step for mapping the output message part to the pipeline.


The record that SAP BC Developer generated for the input or output message already exists in the server namespace. SAP BC Developer will not generate a duplicate record.

[B2BCORE.0092.9030] Error: Input does not have a valid message reference, Web Service Connector was not created.

Within the `<operation>` element, the `message` attribute for the `<input>` element does not reference a `<message>` element within the WSDL. SAP BC Developer cannot find the message specified by the `message` attribute, or the `message` attribute does not have a value. SAP BC Developer does not generate a Web Service Connector for the `<operation>`.

Note: The `message` attribute value must be a QName.

[B2BCORE.0092.9031] Error: Output does not have a valid message reference, Web Service Connector was not created.
Within the <operation> element, the message attribute for the <output> element does not reference a <message> element within the WSDL. SAP BC Developer cannot find the message specified by the message attribute or the message attribute does not have a value. SAP BC Developer does not generate a Web Service Connector for the <operation>.

**Note:** The message attribute value must be a QName.

[B2BCORE.0092.9032] Error: Invalid schema definition for Input signature. Web Service Connector was not created.

The XML Schema definition that contains element declarations or type definitions for the <part> elements in the input message is invalid. Alternatively, the XML Schema definition does not contain the element declarations or type definitions referenced by the <part> elements. SAP BC Developer does not generate a Web Service Connector for the <operation>.

**Note:** This error message is usually accompanied by specific SAP BC schema generation errors. For more information about errors that occur when generating an SAP BC schema from an XML Schema, see “SAP BC Schema Generation Errors and Warnings” on page 643.

[B2BCORE.0092.9033] Error: Invalid schema definition for Output signature. Web Service Connector was not created.

The XML Schema definition that contains element declarations or type definitions for the <part> elements in the output message is invalid. Alternatively, the XML Schema definition does not contain the element declarations or type definitions referenced by the <part> elements. SAP BC Developer does not generate a Web Service Connector for the <operation>.

**Note:** This error message is usually accompanied by specific SAP BC schema generation errors. For more information about errors that occur when generating an SAP BC schema from an XML Schema, see “SAP BC Schema Generation Errors and Warnings” on page 643.
[B2BCORE.0092.9034] Warning: Found port with an invalid binding reference, the port was not generated.

In the WSDL document, a `<port>` element contains a binding attribute that references a `<binding>` that does not exist within the WSDL document. SAP BC Developer cannot find the `<binding>` element specified by the binding attribute.

SAP BC Developer generates the Web Service Connector, but does not generate a MAP step for this port. The `portName` MAP steps set the binding and address information for a port. In a Web Service Connector, the BRANCH on `/port` step contains a child `portName` MAP step for each `<port>` associated with the `<operation>`.

**Note:** The binding attribute value must be a QName.


In the WSDL document, a `<service>` element contains no `<port>` elements. SAP BC Developer generates a Web Service Connector, but if the WSDL document does not contain any `<service>` elements that provide port and address information for an operation, the Web Service Connector will be incomplete.

[B2BCORE.0092.9036] Error: Could not process document. Found binding with an invalid PortType reference, no Web Service Connectors were created.

In a `<binding>` element, the type attribute specifies a `<portType>` that does not exist in the WSDL document. SAP BC Developer does not generate a Web Service Connector.

**Note:** The type attribute value must be a QName.

[B2BCORE.0092.9037] Error: HTTP Binding input type could not be found. Binding was not generated.

SAP BC Developer cannot determine the input MIME type for the HTTP binding.

SAP BC Developer generates the Web Service Connector, but the Web Service Connector does not contain a SEQUENCE step that corresponds to this binding. (In the Web Service Connector, the BRANCH on `/binding` step contains a child SEQUENCE step for each unique binding associated with an operation.)

[B2BCORE.0092.9138] Error: Unknown binding style was found, binding was not created.

The `<binding>` element specifies a protocol other than SOAP, HTTP, or MIME.

SAP BC Developer generates the Web Service Connector, but the Web Service Connector does not contain a SEQUENCE step that corresponds to this binding. (In the Web Service Connector, the BRANCH on `/binding` step contains a child SEQUENCE step for each unique binding associated with an operation.)
Appendix I  WSDL Errors and Warnings

[B2BCORE.0092.9039] Error: URL encoding does not support input variables other than strings and string lists.

The input parameters declared on the Input/Output tab contain a variable that is not a String or a String List. When URL encoding is specified as the input format for the HTTP POST or HTTP GET protocols, SAP BC Server uses the input parameters declared on the Input/Output tab of the service to construct the input message for the WSDL document. For URL encoding, the input signature can contain only String and String List variables. The input signature should not contain Record, Record List, Objects, Object Lists, or String Table variables because these variables cannot be represented in name=value pairs in the HTTP request.

[B2BCORE.0092.9040] Error: PortType name is zero length, Web Service Connector was not created.

In the WSDL document, the <portType> element carries the name attribute, but the name attribute does not have a value. SAP BC Developer does not generate a Web Service Connector.

[B2BCORE.0092.9041] Error: Operation name is zero length, Web Service Connector was not created.

In the WSDL document, the <operation> element carries the name attribute, but the name attribute does not have a value. SAP BC Developer does not generate a Web Service Connector.


When you generate a WSDL document and specify HTTP POST or HTTP GET as the protocol, you can select a record to describe the format of the XML document expected by or produced by the service. The record cannot contain a String Table variable because multi-dimensional arrays cannot be represented in an XML Schema. (SAP BC Developer generates an XML Schema to define types for variables in the record.) To resolve this error, remove the String Table variable from the record, or select a different record.


The XML Schema definition that defines the types and elements for the input and/or output message parts is invalid. The schema errors are listed. For information about errors that occur when SAP BC Server processes an XML Schema definition, see “SAP BC Schema Generation Errors and Warnings” on page 643.


The file you selected to generate the Web Service Connector from is not a .wsdl or .wsd file. You can only generate Web Service Connectors from files with a .wsdl or .wsd file name extension.

SAP BC Developer generated the Web Service Connector successfully, however warnings occurred. Click the Details button to view the warnings.

[BTLINTG.0012.0012] Error occurred while creating Web Service Connector.

SAP BC Developer did not generate a Web Service Connector. Click the Details button to view the errors.
Symbols

! 599
!= 597
$xmldata 373
& 600
&& 600
*, after ACL names 94
< 598
<> 597
>= 598
| 599
|| 599

A

Access Control Lists (ACLs). See ACLs
ACLs
  assigning to folders 95
  assigning to services 95
  checking for services 93
  checking on internal invokes 95
  definition of 93
  inheriting 94
  Internal ACL, description of 95
  removing from folders 97
requirements 26
addBodyPart service 472, 475, 479, 481
adding
  folders 76
  packages 57
  transformers 179
  variables to pipeline map 176
addMimeHeader service 472, 479
address element, in WSDL document 348
addressing
  document nodes 401

pipeline variables 601
adjusting the pipeline 153
After 310
after service for WebTap 511
alarm events
  building handlers for 536
  description of 524, 535
  reasons generated 535
  uses of 535
all content model 193
and operator 600
annotating source code for jcode 301
anonymous type definitions 193
any attribute declaration 193
any element declaration 192
anyURI content constraint 612
API
  for Java services 295
  related documentation 22
applying
  conditions to links 169
  conditions to maps 169
  constraints to variables 211, 612
areas
  behavior and operation 34, 35
  general layout 28
  resizing 35
  Service Browser 29
  zooming 35
arithmetic services 179
array variables
  as SOAP Arrays 350
default behavior for mapping 580
description of 580
guidelines for mapping 167
mapping to or from array variables 165, 580
mapping to or from scalar variables 165, 580
mapping to transformers 183
representation in XML Schemas 349
assigning
ACLs to folders 95
ACLs to services 95
default value to a variable 173
universal names to services 98
values to the pipeline 172
variable values to a variable 173
version numbers to packages 61
asterisk (*), after ACL names 94
as-user property 97
asynchronous transaction
eexample of (in flow) 460
eexample of (in Java) 457
attribute declaration 192
attribute reference 192
attributes 412, 421
NAME attribute 416
pattern matching 416
audit events
audit.log file 540
building handlers for 538
default audit handler 540
description of 524, 536
setting audit levels 537
Audit Level property 537
audit.log file, for audit events 540
authenticating S/MIME messages 468, 497

B
base64Binary content constraint 613
before service for WebTap 511
binary content constraint. See base64Binary or hexBinary
binding element, in WSDL document 337, 347
binding options for COM objects 313
blue links in the Pipeline Editor 167
boolean content constraint 613
Boolean expressions, using in queries 425
BR tag 418
BRANCH step
adding steps to targets 143
behavior in flow diagram view 142
branching
on expressions 113

on switch value 110
creating 116
creating a multi-step child for 115
definition of 72, 554
evaluate-label property 113
specifying
the default step 114
the label value 111
the switch variable 111
switch value 110
using conditional expressions 113
using in a flow 110
using SEQUENCE as a target of 115
using with regular expressions 111
breakpoints
and the Trace Into command 267
clearing from flow steps 267
clearing from transformers 268
in flow services
clearing 267
locating 269
setting 267
listing all 269
overview 266
point when processing halts 267, 268
removing 269
setting on a flow step 267
setting on transformers 268
browser clients
creating 329
invoking database services from 444
invoking services from 330, 331
testing from 257
WebTap 520
Browser Recorder
for load and query services 430
using 431
build_EncryptedSMIME sample service 487
build_MultipartMIME sample service 480
build_SignedAndEncryptedSMIME sample service 489
build_SignedSMIME sample service 484
build_SimpleMIME sample service 478
building
event handler sample 533
  event handlers 533
  flow services 70
built-in services
  for arithmetic operations 179
  for date/time transformations 179
  for node validation 229
  for pipeline validation 222
  for record validation 229
  for remote services 108
  for string manipulation 179
  for XML validation 219
  invoking 108
  pub.schema:validate 219, 229
  pub.schema:validatePipeline 222
byte content constraint 613

C
C/C++ clients
  creating 322, 323
  creating a make file 324
  for database services 445
C/C++ services
  accessing databases through 442
  compiling with a make file 306
  creating 306, 307
  caching
    definition of 88
    elements to improve Developer performance 50
    services not suited for 89
    services suited for 88
    setting 90
    using prefetch 90
  call stack 256
  CDATA content constraint. See normalizedString content constraint
Certificate Authorities 467, 483, 484, 497
certificate chains 467, 483, 484, 497
certificates. See digital certificates
changing
  level of a step 104
  passwords 48
  position of an step 104
  step layout in flow diagram view 147
character ranges 419
checking, ACLs for services 93
child flows
  stepping in/out of child flows 265
  tracing 263
children of a flow step 104
choice content model 193
circular dependencies 64
class files
  location of 305
class files, location of 298
classes subdirectory 305
clearing
  breakpoints in flow steps 267
  breakpoints on transformers 268
clear-signing, S/MIME messages 468
client applications
  creating browser-based 329
  creating C/C++ 322
  creating Excel 327
  creating Java 320
  creating Visual Basic 324
  for database services 444, 445
closing a session on SAP BC Server 47
coded services 288
collapse white space 624
collapsing transformers 185
COM objects
  binding options 313
  invoking as services 311
  using with SAP BC Platform 310
COM services
  creating 310
  invoking 313, 315
combining Trace and Step commands 261
commands
  Disable Step 269, 271
  Enable Step 269, 271
  Load Pipeline Locally 278
  Reset 261
  Save Pipeline from Server 278
  Save Pipeline Locally 276
Index

Save Pipeline to Server 276
Set Breakpoint 267
Step 260, 264, 265
Step Into 260, 264, 265
Step Out 264, 265
Trace 260, 262
Trace Into 260, 262, 263
Trace to Here 260, 262, 263
comment property 106
comments for jcode 301
comparison operators
   using for expressions 597
   using in queries 425
compiling
   C/C++ services 310
   Java services 302
   services with Developer 296
   services with jcode 302, 304
   Visual Basic services 313
complex type definition 193
   anonymous 193
   empty content 194
composite mode (jcode) 303
conditional expressions
   logical operators 599
   relational operators 597
   syntax 594
   using with the BRANCH step 113
conditional maps, disabling 273
Conditional tab. See General tab
conditions
   applying to links 169
   applying to maps 169
   disabling on maps 273
connecting to SAP BC Server 27, 47
constraining facets
   applying 215
   definition of 211
   described 623
   for pipeline validation 612
content constraints 190
   applying to variables 211
   constraining facets 211, 623
   description of 211
   description of 190, 612
   editing for simple types 198
   for validation 612
   for XML validation 190
   ignoring during validation 221, 231
content models
   all 193
   choice 193
   empty 194
   mixed 194
   sequence 193
content types
   applying to variables 213
   customizing 213
   specifying constraining facet values 215
Content-Transfer-Encoding header field 475, 476, 479, 481
Content-Type
   MIME messages 466
   S/MIME messages 468, 469, 471
   conventions used in this document 20, 21
   converting string lists to record lists 164
copying
   by reference, description of 160
   by value, description of 160
   elements 36
   elements from the Results tab 255
   pipeline values 174
   set values 174
   transformers 185
createEncryptedData service 472, 487, 488
createMimeData service 472, 473, 479, 481, 493, 494, 495, 498, 500, 503
createSignedAndEncryptedData service 472, 489, 490
createSignedData service 472, 484, 485
for IS schemas 190
for XML validation 190
structural 190
viewing for variables 216
viewing for simple types 198
for validation 612
for XML validation 190
ignoring during validation 221, 231
creating
  code for Java services 296
  event filters 529
  event filters for services 531
  event handler sample 533
  event handlers 533
  flow services 75, 76
  folders 76, 306
  IS schemas 195
  Java services with an IDE 301
  Java services with C/C++ 306
  Java services with Developer 291, 295
  Java services with Visual Basic 310, 312
  Java services with your own IDE 298
  links to array variables 580
  links to record variables 163
  packages 57
  specifications for C/C++ services 306
  version numbers for packages 61
Web Service Connectors 366
WSDL documents
  default.xsd 344
  describing service signature 353
  files produced 344
  HTTP GET protocol 359
  HTTP POST protocol 359
  namespaces.dtd 345
  prefix.xsd 344
  process overview 340
  SOAP Message protocol 352
  SOAP RPC protocol 341
  ST#.xsd 345
  URL encoding restrictions 359
cursors 289
customizing, content types 213

D
data transformations, defined 150
data types
  IData objects 580
  mapping with Pipeline Editor 163
  Object 34
  Object List 34
  Record 33
  Record List 33
  Record Reference 34
  Record Reference List 34
  representation in XML Schemas 348
  String 33
  String List 33
  supported by SAP BC Server 578
tables 579
  Values objects 580
data validation
  allowing undeclared variables 213
  applying constraints 211
  benefits of 210
  blueprints for 210
  content constraints 211
description of 210
  errors 628
  exceptions 233, 639
  input/output validation
    description of 226
    performing via Input/Output tab 227
    performing via INVOKE step properties 228
  node validation
    description of 229
    performing 229
    performing
      input/output validation 226
      node validation 229
      pipeline validation 222
      record validation 229
    XML validation 219
  pipeline validation
    description of 222
    performing 222
  record validation
    description of 229
    performing 229
    requiring variable existence 212
  structural constraints 211
types of 210
  XML validation
    description of 218
Index

performing 219
database services
  converting database rows to IData 444
  creating 434
  creating flow services 435
  creating from a table 439, 440
  creating from SQL 436, 438
  creating in Java, C/C++ or VB 442
  error handling 444
  invoking from a browser 444
  invoking from a client application 445
date content constraint 613
date conversion services 179
dateTime content constraint 614
DCOM objects
  invoking as services 311
  using with SAP BC Server 310
debug level, setting on server 281
debugging flow services
  breakpoints 266
  combining trace and step 261
  disabling a single flow step 269
  disabling a transformer 271
  disabling conditions on maps (links) 273
  dumping the pipeline to server.log 283
  enabling a single flow step 269
  enabling a transformer 271
  modifying the pipeline 274
  overview 248, 259
  resetting debug mode 261
  resolving service and record relationships 236
  stepping into a child flow 265
  stepping into a transformer 266
  stepping through a flow 260, 264
  tracing a flow 260, 262
  tracing into a child flow 263
  using server debug facility 280, 281
debugging Java services
  class file location 305
  general steps 304
  generating the class file 304
  registering the class file 305
  resolving service and record relationships 236
debugLog service 282
decimal content constraint 614
declaring
  input and output parameters 81
  output for a service 84
Default ACL, description of 97
default event handler
  audit 540
  exception 541
  GD End 544
  GD Start 544
  session end 548
  session expire 549
  session start 547
  stat 550
  Tx End 552
  Tx Start 552
Default package 54
default SOAP processor, using as directive for WSDL document 353
default value, assigning to a pipeline variable 173
default.xsd, about 344
defining packages 57
definitions element, in WSDL document 338, 346
deleting
  breakpoints on transformers 268
  elements 37
  event subscriptions 532
  links between variables 169
  packages 59
dependencies
  identifying for packages 63
  removing for packages 64
dependents
  checking when renaming IS elements 242
  description of 236
  finding service and record 240
DER format 483, 487
describing service signature 353
detached signature, S/MIME messages 468
Developer. See SAP BC Developer
diagram view. See flow diagram view
digest of S/MIME message 468
digital certificates
  authenticating 467, 468, 497
  Certificate Authorities 467
  certificate chains 467
  expired certificates 497
  getting from partners 471
  inability to read 497
  invalid chains 497
  overview of 467
  required to decrypt a message 500
  required to encrypt a message 487
  required to sign a message 483
  required to verify a signature 498
  storing locally 471
  trusting 468, 497
  X.509 format 472
digital signatures 467, 483
  authenticating 468, 497
  clear-signing 468
  detached signature 468
  explicit 468
  implicit 468, 469
  overview of 468
  trusting 468, 497
  verification failures 497
  verification process 497
dimensionality, defined 183
directories
  Java services 298
Disable Step command 269, 271
disabling
  a condition placed on a map (link) 273
  a flow step 269
  a transformer 271
dispatch services
  for COM 311
  for DCOM 311
Document objects
  as output from loadDocument 383
  definition of 383
  node 383
  parsing 382
Document Type Definitions. See DTDs
Document View tab 403
documentation
  accessing for packages 58
  conventions used 20
  printing 23
  related manuals 22
  using effectively 20
  viewing 23
documenting packages 58
documents
  parsing 382
  retrieving from Web 383
double content constraint 614
DROP modifier 153, 175
drop zones 140
dropping a value from the pipeline 175
DTDs
  creating IS schemas from 195
  creating records from 200
  IS schema generation warnings 643
duration content constraint 615
E
early binding, for a Visual Basic service 313, 315
ing editing
  event subscriptions 531
  flow services 70
  service property 107
  simple types in IS schema 198
Editor area 31
element declaration 192
element reference 192
elements (IS)
  caching to improve Developer performance 50
  copying 36
  creating 36
  deleting 37
  dependents 236
  description of 236
  displayed in Service Browser 29
  exporting 60
  finding in Service Browser 238
  locking 36
Index

moving 36
references 237
renaming 37, 242
saving 40
unlocking 36
elements, in a WSDL document 336, 346
emailing XML to a service 378
empty content 194
Enable Step command 269, 271
enabling
    flow steps 269
    transformers 271
encryption of MIME messages 467, 470
end-to-end mapping 178
Enforce ACL on Internal Invokes option 95
entering input for a service 250
ENTITIES content constraint 615
ENTITY content constraint 615
enumeration constraining facet 215, 623
error codes, digital signature failures 497
error handling
    in database services 444
    in flow services 125
    in guaranteed delivery 453
    in loadDocument 399
    in queryDocument 427
error.log file, for exception events 541
errors
    flow service generation 643
    IS schema generation 643
    record generation 643
    when generating Web Service Connectors 648
    when performing data validation 628
    when performing pipeline validation 628
evaluate-label property 117
event filters
    creating 529
    creating for services 531
event handlers
    See also Event Manager
creating 533
creating sample 533
description of 525
for alarm events 536
for audit events 538
for exception events 540
for GD End events 544
for GD Start events 543
for port status events 545
for replication events 546
for session end events 548
for session expire events 548
for session start events 547
for stat events 549
for Tx End events 551
for Tx Start events 551
stages of creating 533
Event Manager
    See also event filters, event handlers, events
    alarm events
        building handlers 536
description of 524, 535
    uses 535
    audit events
        building handlers 538
default handler 540
description of 524, 536
    setting audit levels 537
description of 524
    event behavior 525
event handlers 525
    Event Manager command 526
    exception events 524, 540
        building handlers 540
default handler 541
    filters 529
    GD End events 542
        building handlers 544
default handler 544
    uses 542
    GD Start events 542
        building handlers 543
default handler 544
    uses 542
guaranteed delivery events 524
    pattern strings 529
port status events 524, 545
  building handlers 545
  uses 545
replication events 524, 546
  building handlers 546
  uses 546
run-time behavior 525
session end events 547
  building handlers 548
  default handler 548
session events 524, 546
  uses 547
session expire events 547
  building handlers 548
  default handler 549
session start events 547
  building handlers 547
  default handler 547
stat events 524, 549
  building handlers 549
  default handler 550
  uses 549
subscriptions
  creating 526
  creating filters for 529
  deleting 532
  editing 531
  managing 526
  suspending 532
  viewing 531
transaction events 525, 550
Tx End events 551
  building handlers 551
  default handler 552
  uses 551
Tx Start events 551
  building handlers 551
  default handler 552
  uses 551
viewing subscriptions 532
events
  See also event handlers, Event Manager
  creating filters 529
creating filters for services 531
deleting subscriptions to 532
ingoing subscriptions to 531
guaranteed delivery 542
subscribing to 526
suspending subscriptions to 532
types of 524
viewing subscriptions to 531
Excel clients, creating 327
exception events
  building handlers for 540
  default audit handler 541
description of 524, 540
  error.log file 541
exceptions
during a test 255
  for data validation 233, 639
executing services from Developer 249
existing record. See record reference
EXIT step
  creating 131
definition of 73, 557
  using in a flow 130
exit-on property 125
expanding transformers 185
expired certificate error 497
explicit mapping, description of 156
explicit signatures, S/MIME messages 468
explicit universal name
  use in WSDL document 345
exporting, elements and packages 60
expressions
  addressing pipeline variables in 601
  branching on 113
  operators for 597
  syntax for 594
  using for pipeline mapping 169
extending the Java class 294
extends (Java keyword) 294
external records. See records
external specification. See specification
externally invoked services, description of 93
extract_EncryptedSMIME sample service 501
extract_MultipartMIME sample service 495
extract_SignedAndEncryptedSMIME sample service 503
extract_SignedSMIME sample service 499
extract_SimpleMIME sample service 494

F
facets, See constraining facets.
failIfInvalid property 221, 223, 231, 639
failure
  of a flow step 125
  of a REPEAT step 120
filtering events 529
filters. See event filters
finding
  elements in Service Browser 238
  invoked services 110
  service and record dependents 240
  service and record references 242
  unresolved pipeline references 244
finding invoked services 110
float content constraint 615
flow control steps
  BRANCH 72, 110
  definition of 71
  EXIT 73, 130
  LOOP 72, 126
  REPEAT 72, 118
  SEQUENCE 73, 125
flow diagram view
  about 134
  advantages of 135
  BRANCH step children 142
  changing layout of steps 147
  creating flow services in 140
  drop zones 140
  elements in 136
  inserting steps in 140
  moving steps logically 147
  properties shown 139
  redrawing flow service 148
  relocating steps 147
  resetting layout 148
  step properties 139
  switching to 140
Flow Editor
  definition of 102
  inserting steps into 103, 140
  switching views 140
flow operations. See flow steps
flow services
  See also services
  ACL checking 93
  appearance in flow diagram view 136
  assigning ACLs 95
  breakpoints 266, 267
  caching to improve Developer performance 50
  changing appearance in flow diagram view 147
  creating 76
  creating in flow diagram view 140
  debugging 259
  definition of 70
  disabling a single step 269
  disabling a transformer 271
  disabling conditions placed maps 273
  editing 70
  enabling a single step 269
  enabling a transformer 271
  enforcing ACL checks internally 95
  errors creating from DTD or XML Schema 643
  externally invoked 93
  inheriting ACLs 94
  inserting steps in flow diagram view 140
  inserting steps in flow tree view 103
  internally invoked 93
  printing 99
  redrawing in flow diagram view 148
  relationship to WIDL 71, 382
  removing ACLs 97
  resetting layout in flow diagram view 148
  resolving service and record relationships 236
  response to failure 125
  security 93
  stages of creating 75
  stepping in/out of child flows 265
  stepping in/out of transformers 266
  stepping through 264
tracing 262
viewing in browser 99
flow steps
BRANCH 72, 110
changing the level of 104
changing the position of 104
definition of 71
disabling 269
enabling 269
EXIT 73, 130
failure of 125
grouping 125
hierarchical order of 104
inserting in flow diagram view 140
inserting into a flow service 102
INVOKE 72, 107
LOOP 72, 126
MAP 72, 132
moving
in flow diagram view 147
in flow tree view 104
parent/child relationships 104
properties 105, 106
re-enabling 269
relocating in flow diagram view 147
REPEAT 72, 118
SEQUENCE 73, 125
setting breakpoints on 267
viewing in flow diagram view 140
flow tree view, switching to 140
folders
assigning ACLs 95
creating 76
inheriting ACLs 94
removing ACLs 97
fractionDigits constraining facet 215
frag mode (jcode) 303
FTPing XML to a service 376

g
GD End events
building handlers for 544
default handler 544
description of 542
pub.remote.gd:end 542
uses of 542
when generated 542
GD Start events
building handlers for 543
default handler 544
description of 542
pub.remote.gd:start 542
uses of 542
when generated 542
gDay content constraint 616
General tab 171
Generate Code command 321, 323, 325
Generate WSD command 341, 355, 361
generating
WSDL documents
default.xsd 344
describing service signature 353
files produced 344
HTTP GET protocol 359
HTTP POST protocol 359
namespaces.dtd 345
prefix.xsd 344
process overview 340
SOAP Message protocol 352
SOAP RPC protocol 341
ST#.xsd 345
URL encoding restrictions 359
getBodyPartContent service 473, 492, 494, 495, 499, 501, 502,
503
ggetBodyPartHeader service 473, 492
gGetContentType service 473
gGetEnvelopeStream service 472, 478, 479, 481
gGetMimeHeader service 473, 492
gGetNumParts service 473, 495, 502
global declarations 192
gMonth content constraint 616
gMonthDay content constraint 616
Go To Breakpoint 269
Go To command 110
gray links 155
grouping flow steps 125
guaranteed delivery
  creating client applications for 454
definition of 450
ending a transaction 454
error handling 453
generating guaranteed delivery (GD) events 542
generating transaction (Tx) events 542
heuristic errors 453
in a flow service
  checking for results 460
  ending a transaction 459, 461
  invoking a service 459
  making a connection 459, 460
  retrieving results 460
  sample flow, asynchronous 460
  sample flow, synchronous 459
  starting a transaction 459, 460
  submitting a request 460
in a Java client
  checking for results 457
  ending a transaction 456
  invoking a service 456
  making a connection 456
  retrieving results 457
  sample code, asynchronous 454, 457
  starting a transaction 456
  submitting a request 457
  invoking a service 454
  Job Manager 450
  making a connection 454
  sample code
    asynchronous 457
    synchronous 454
sample flow
  asynchronous 460
  synchronous 459
server properties 451
starting a transaction 454
starting Job Manager 455
stopping Job Manager 456
time-to-live (TTL) 452
guaranteed delivery events
  See also GD End events, GD Start events
  compared to transaction events 542
description of 524, 542
overview 542
txout.log file 544
when generated 542
guidelines
  for assigning startup, shutdown, and replication services 66
  for mapping array variables 167
  for mapping record variables 163
  for mapping variables 158
  for mapping variables of different data types 163
gYear content constraint 617
gYearMonth content constraint 617
H
hailmary mode (jcode) 304
hard-coding input variables 172
header fields, adding to MIME messages 475
help system for Developer 51
heuristic errors 453
hexBinary content constraint 617
HTML documents
  extracting information from 400
  retrieving from Web 383
HTTP GET protocol
  service requirement for generating WSDL documents 360
  service requirements for generating WSDL documents 360
  URL encoding restrictions 359
HTTP POST protocol
  service requirements for generating WSDL documents 360
  specifying for WSDL documents 359
  URL encoding restrictions 359
I
ID content constraint 617
IData objects
  compared to Values 580
  creating 289
description of 288
  getting data from 289
  positioning cursors 289
  putting data in 289
  using in a Java service 299
IDE
assigning a super class with 294
creating source code in 292
defining private methods with 294
implementing interfaces with 294
importing Java packages 294
in Developer 290, 292
using other IDEs 298
using the Shared tab 293
using the Source tab 292
identifying
package dependencies 63
replication services 67
shutdown services 67
startup services 67
IDREF content constraint 618
IDREFS content constraint 618
ignoreContent property 221, 231
implementing interface in Java services 294
implements (Java keyword) 294
implicit mapping
description of 155
for MAP steps 178
implicit signatures, S/MIME messages 468, 469
implicit universal name
use in WSDL documents 345
import (Java keyword) 294, 299
import mechanism, in XML Schemas 197
importing Java packages 294, 299
in-array property 127
include mechanism, in XML Schemas 197
index values
for all members 412, 422
for multiple members 412, 422
for single member 412
specifying with pattern match 414, 424
Indexing tab 167
inheriting ACLs 94
input element, in WSDL document 347
input signature, describing for WSDL documents 353, 360
input variables
declaring for a service 81, 82
declaring in a record 200
entering test values for 250
hard-coding 172
loading from a file 252
mapping considerations 158
mapping in the pipeline 156
saving values to a file 251
input/output parameters
declaring for a service 81
generating Java code from 296
Input/Output tab 84
input/output validation
description of 226
performing via Input/Output tab 227
performing via INVOKE step 228
InputStreams, behavior in flow services 496
inserting
INVOKE step 108
steps in BRANCH targets 143
steps into flow services 103, 140
transformers 179
installing Java services 298
int content constraint 618
integer content constraint 618
Integrated Development Environment. See IDE
Integration Server. See SAP BC Server
interfaces, implementing in Java service 294
Internal ACL, description of 95
internally invoked services
description of 93
enforcing ACL checks 95
INVOKE step 110
as-user property 97
creating 108
definition of 72, 559
invoking built-in services 108
invoking services on another server 108
performing validation via step properties 228
Pipeline Editor 151
service property 107
using in a flow 107
invoking
built-in services 108
database services from a browser 444, 445
remote services 108
services 108
services from a browser 330, 331
services on another server 108
Visual Basic services 313
WebTap services 520
IS elements
  caching 50
  copying 36
  creating 36
  deleting 37
  dependents 236
  displayed in Service Browser 29
  finding dependents 242
  moving 36
  references 237
  renaming 37
  saving 40
IS schemas
  appearance 190
  constraints 190
  content constraints 190
  creating 195
  description of 190
  editing simple types 198
  errors from generating 643
  generating multiple 197
  import element 197
  include element 197
  structural constraints 190
  using to validate a node 229
  using to validate an XML document 218
  warnings from generating 643

J
Java clients
  creating 320, 321
  for database services 445
Java keywords
  extends 294
  implements 294
  import 294, 299
Java services
  accessing databases through 442
  adding private methods to 294
  compiling with jcode 302
  creating 295, 296
  creating with an IDE 301
  creating with Developer 291
  creating with your own IDE 298
  debugging 304
  extending the class of 294
  implementing interface in 294
  importing packages into 294, 299
  installing manually 298
  jcode requirements 301
  location of class files 298
  location of source files 299
  publishing to the server 298
  required code 299
  setting run-time options 297
  stages of development 291
  structure of 290
  using with wSAP BC Platform 290
  validating pipeline from 224
jcode utility
  composite mode 303
  frag mode 303
  hailmary mode 304
  make mode 302
  purpose of 298
  source code samples 604
  source code tags 301
  update mode 304
Job Manager 450

L
label property
  for evaluating expressions 113
  for general use 106
  for specifying the default BRANCH step 114
  for targeting BRANCH steps 111
language content constraint 618
late binding, for a Visual Basic service 313
length constraining facet 215, 623
libs directory 306, 310, 313
lines, extracting from a document 418
links
  blue colored 167
deleting 169
disabling conditions 273
execting at run time 160
gray colored 155
in Pipeline Editor 155
load and query services
  creating 384, 404, 429, 430
  creating from URL 428
  creating with Browser Recorder 430
  definition of 382
loadDocument 383
queryDocument 400
relationship to WIDL 382
shortcuts for 427
Load Pipeline Locally command 278
loadDocument
  creating from URL 428
  creating with Browser Recorder 430
data variable 388
encoding variable 395
error handling 399
expandDTD variable 396
failOnHTTPError variable 398
headers variable 393
input 385
isXML variable 396
loadAs variable 397
method variable 386
output 398
shortcuts 427
url variable 385
using 384, 429
loading
  input values from a file 252
  pipeline values from a file
    into Results tab 278
    overview 278
    with restorePipelineFromFile service 279
  local names 98
  locating a breakpoint 269
log files
  audit.log and audit events 540
  error.log and exception events 541
  session.log and session end events 548
  session.log and session expire events 549
  session.log and session start events 547
  stats.log and stat events 550
  txin.log and transaction events 552
  txout.log and guaranteed delivery events 544
logging on to SAP BC Server 27, 47
logical operators, for conditional expressions 599
logToFile service
  audit 540
  exceptions 541
  GD End 544
  GD Start 544
  session end events 548
  session expire events 549
  session start events 547
  stat 550
  Tx End 552
  Tx Start 552
long content constraint 618
LOOP step
  creating 129
  creating nested loops 126
  definition of 72, 561
  setting in-array 127
  setting out-array 128
  using in a flow 126
M
make file
  creating 306
  using to compile C/C++ services 310
make mode (jcode) 302
MAP modifier 153, 156, 157, 163
executing at run time 160
MAP step
  debugging 266
  definition of 72, 563
  disabling transformers 271
  enabling transformers 271
extracting data from  472, 473, 492
extracting data from encrypted messages  500
extracting data from multipart messages  495
extracting data from signed and encrypted messages  502
extracting data from signed messages  496
extracting header field from  473, 492
extracting specific body part  494
generating a message stream  473, 478, 480
header fields  465, 474, 475
message headers  466, 474, 475
MIME entities  473
MIME Object  473
MIME object  473, 492
MIME_Version header field  465, 475
mimeData Object  473, 474, 479, 492, 493, 494, 495
multipart messages  466, 480, 482, 495
number of parts, discovering  473, 495
order of parts  482
part headers  466, 475, 476
payload  465
receiving  492
services  472
signing  483
signing and encrypting  489
testing for encryption  500
testing for signature  496, 498
mimeData Object  473, 474, 479, 491, 492, 493, 494, 495
minExclusive constraining facet  215, 624
minInclusive constraining facet  215, 624
minLength constraining facet  215, 624
mixed content  194
modifying the pipeline  153
month content constraint. See gMonth content constraint
  moving
  elements  36
  steps in flow diagram view  147
  steps in tree view  104
  steps within a flow  104
Multipurpose Internet Mail Extensions. See MIME

N
Name content constraint  619
name transformations
definition of  150
namespace information (for services)
  creating source file from  303, 304
  location of  298
  selecting in queries  425
  updating with jcode  303
namespace names  98
namespaces  345
  in WSDL documents  339
  usage in universal names  98
namespaces.dtd, about  345
NCName content constraint  619
negativeInteger content constraint  619
NMTOKEN content constraint  619
NMTOKENS content constraint  619
nodes
  addressing elements in  401
definition of  383
  extracting information from  401
  passing to queryDocument  400
nonNegativeInteger content constraint  619
nonPositiveInteger content constraint  620
nonrepudiation of S/MIME messages  468
not operator  599
NOTATION content constraint  622
notification, of server shutdown  48
ns directory  298

O
Object data type
definition of  34, 579
  representation in XML Schemas  350, 352
Object List data type
definition of  34, 579
  representation in XML Schemas  350, 352
object properties, definition of  411
object references
  definitions of  572
  sibling operators  574
online help  51
opening a session on SAP BC Server  47
operation element, in WSDL document  337, 347
operations. See flow steps
operators
  for conditional expressions 597
  logical 599
  relational 597
optional variables for validation 212
or operator 599
out-array property 128
output element, in WSDL document 347
output signature, describing for WSDL documents 353, 360
output templates
  assigning to a service 91
  definition of 91
output variables
  declaring 84
  declaring for a service 81
  declaring in a record 200
  mapping considerations 158
  mapping in the pipeline 157
Overwrite Pipeline Value option 173

P
packages
  accessing documentation 58
  adding 57
  assigning version numbers 61
  creating 57
  creating circular dependencies 64
Default 54
  definition of 54
  deleting 59
  dependencies, using with startup services 65
  description of 54
  documenting 58
  exporting 60
  guidelines for creating 57
  identifying dependencies 63
  importing into Java services 294, 299
reloading 59
  reloading vs refreshing 59
  removing dependencies 64
  required by Java services 299
  sample services 54
  viewing patch history 62
WmDB 54
WmPartners 54
WmPublic 54
WmWin32 55
parameters
  applying constraints 211
  benefits of declaring 82
  declaring 78, 80, 81, 86
  declaring for a service 82, 84
parent/child relationships in a flow 104
parsing documents 382
part element, in WSDL document 337, 346, 347
passing XML to a service 372
passwords
  changing 48
  requirements 49
patch history
  removal by Integration Server 62
  viewing for a package 62
pattern constraining facet 215, 624
pattern matching
  in event subscriptions 529
  in queries 414, 424
PDF, viewing 23
Perform Variable Substitution option 173
performing
  data validation
    benefits of 210
    blueprints for 210
    description of 210
    input/output validation 226
    node validation 229
    pipeline validation 222
    record validation 229
    types of 210
    XML documents 219
input/output validation 226
node validation 229
pipeline validation 222
record validation 229
XML validation 218
pipeline
  adding variables to 176
addressing variables 601
adjusting 153
assigning values to 172
changing values 274
checking for variables 596
conditional mapping 169
copying values 174
definition of 73
DROP modifier 175
dropping values 274
dropping variables from 175
inspecting modifiers 244
MAP step 132
mapping 159
array variables 165
array variables, default rules 580
input variables 156
output variables 157
scalar variables, default rules 580
transformers 181
variables of different data types 163
mapping array variables, guidelines for 167
modifying 274
Overwrite Pipeline Value option 173
Perform Variable Substitution option 173
Pipeline In 151
Pipeline Out 152
restoring from a file 278
saving 275
saving from Results tab 276
saving with savePipelineToFile service 277
Service In 151
Service Out 152
SET VALUE modifier 172
stages of 151
validating 222
validating via built-in services 222
validating via Java service 224
viewing after run-time exception 257
viewing with Developer 253
Pipeline Editor 150
blue links 167
copying by reference 160
copying by value 160
default behavior for arrays 168
default mapping rules 580
deleting links 169
deleting maps between variables 169
for a MAP step 152
for INVOKE step 151
inserting transformers 179
links 155
mapping
array variables 165
array variables, default rules 580
array variables, guidelines for 167
record variables 163
scalar variables, default rules 580
transformers 181
variables 155
printing 154
transformer movement 182
viewing in browser 154
Pipeline In 151, 153
pipeline modifiers
description of 153
DROP 153
MAP 153
SET VALUE 153
Pipeline Out 152, 153
pipeline references
description of 237
finding 244
Pipeline tab 150
pipeline validation
description of 222
errors 628
exceptions 639
performing via built-in services 222
via Java service 224
port element, in WSDL document 337, 347
port status events
building handlers for 545
description of 524, 545
uses of 545
portType element, in WSDL document 337, 347
positiveInteger content constraint 620
posting XML to a service 375
PRE tag 418
precision constraining facet 624
prefetch 90
prefix.xsd,about 344
preserve white space 624
printing
  flow services 99
  pipeline editor 154
  records 203
  this guide 23
private keys 467, 470, 483, 484, 485, 500, 501
private methods, defining in Java service 294
processCertsOnlyData service 487
processEncryptedData service 473, 500, 501, 502, 503
processSignedData service 473, 496, 498, 499, 503
programming languages
  creating services with 288
  supported 288
properties 411, 575
  as-user 97
  comment 106
  common to all steps 106
  definition of 105
  evaluate-label 117
  exit-on 125
  for transformers 180
  in-array 127
  label 106
  label (for BRANCH steps) 111, 114
  out-array 128
  repeat-on 119, 120, 566
  scope 116
  service 107, 109, 180
  setting for variables 205
  switch 111, 117
  timeout 117, 121, 122, 129
  validate-in 109, 180
  validate-out 109, 180
Properties tab 105
pub.remote:gd:end 542
pub.remote:gd:start 542
pub.remote:invoke 108
pub.schema:validate service 219, 229
pub.schema:validatePipeline service 222
pub.web
documentToRecord 398
loadDocument 383
queryDocument 400
pub.webtap:triggerSpec 511
Public Key Infrastructure 467
public keys 467, 470, 487
publishing services to the server 298
Q
QName content constraint 622
queries
  attributes 416
  Boolean expressions in 425
  character ranges 419
  comparison operators 425
  creating
    by pointing 409
    manually 407, 408
  definition of 383
  for BR elements 418
  for PRE elements 418
  for tables 414
  masks 416
  namespaces 425
  pattern matching 414
  scope 414
  selecting
    attributes with WQL 412
    attributes with XQL 421
    child elements 424
    lines 418
  WQL properties 411
  XQL methods 419
query and load services 382
queryDocument
  creating from URL 428
  creating with Browser Recorder 430
  error handling 427
  input 400
output 427
shortcuts 427
using 404, 429
using WQL with 401
using XQL with 401

R
rec folder, for Web Service Connector 367
Record data type 33, 578
representation in XML Schemas 349, 351
Record List data type
converting from String List 164
defined 33, 578
representation in XML Schemas 349, 352
Record Reference data type 34, 578
representation in XML Schemas 350, 352
Record Reference List data type 34, 578
representation in XML Schemas 350, 352
record reference list variable 204
record reference variable 204
record validation
description of 229
errors 628
exceptions 639
performing 229
records
applying content constraints 612
assigning
to a Record or Record List 204
to a specification 200
assigning to a Record or Record List 200
benefits of 200
creating 200, 201
errors when creating from DTD or XML Schema 643
finding dependents 242
in pipeline validation 222
mapping 163
printing 203
resolving service and record relationships 236
unresolved links 352
using as service parameters 200, 204
viewing in browser 203
redraw flow services in flow diagram view 148
re-enabling
flow steps 269
transformers 271
references
description of 237
finding service and record 242
inspecting pipeline 244
registering a Java service 305
regular expressions
creating event filters with 531
definition of 586
operators 587
using as a BRANCH label 111
using in a mask 586
using in a WebTap rule 510
using in an Object reference 573, 576
relational operators, for conditional expressions 597
RelayHandler class 507
reloading packages 59
relocating flow steps in flow diagram view 147
remote servers
invoking services on 108
remote services
invoking 108
removing
ACLs from folders and services 97
ACLs from services 97
breakpoints 269
breakpoints from transformers 268
package dependencies 64
packages 59
replication services 68
shutdown services 68
startup services 68
variables from the pipeline 175
renaming
elements 37, 242
services 37
transformers 187
REPEAT step
creating (on failure) 121
creating (on success) 123
Index

definition of  72, 565
failure 120
specifying the repeat condition  119, 120, 566
using in a flow 118
using to retry a failed step 120
using to retry a successful step 123
repeat-on property 119, 120, 566
replace white space 624
replication events
building handlers for  546
description of  524, 546
uses of  546
replication services
definition of  66
guidelines for assigning  66
identifying 67
removing  68
when to use 66
requirements for passwords  49
requiring
ACL checking on internal invokes 96
variable existence for validation 212
Reset command 261
resetting a flow service to default layout 148
resizing window areas 35
restorePipelineFromFile service  276, 278
restoring
pipeline values from a file
into Results tab  278
overview 278
with restorePipelineFromFile service  279
sessions 48
Results tab
copying elements from 255
general information 254
saving results from 275
retrieveDTx 457
retrieveTx 457
retrieving documents from the Web 383
retrying a flow step 118
RSA technology 467
rules engine for WebTap 508
Rules tab 510
Run command 249
Run in Browser command 257
running out of memory, during validation 234
running services from Developer 249
run-time exceptions 255
run-time settings 297
S
S/MIME messages
See also MIME messages
ascertaining identity of sender 468, 497
authentication error codes 497
authentication of 468, 497
build_EncryptedSMIME example 487
build_SignedAndEncryptedSMIME example 489
build_SignedSMIME example 484
Certificate Authorities 467, 483, 484, 497
certificates 467, 471, 483, 487, 489, 497
Content-Type header fields 468, 469, 471
creating 472
creating encrypted messages 487
creating signed and encrypted messages 489
creating signed messages 483
decrypting 500, 502
definition of 467
digital signatures 467, 468, 483
encrypting 467, 470, 487
error codes 497
examples
encrypting a message 487
extracting data from a signed and encrypted message 503
extracting data from a signed message 499
extracting data from an encrypted message 501
signing a message 484
signing and encrypting a message 489
expired certificates 497
explicit signatures 468
extract_EncryptedSMIME example 501
extract_SignedAndEncryptedSMIME example 503
extract_SignedSMIME example 499
extracting data from 472
extracting data from encrypted messages 500
extracting data from signed and encrypted messages 502
Index

extracting data from signed messages 496
header fields 468, 469, 471
implicit signatures 468, 469
invalid certificates 497
message digest 468
message integrity algorithm 468
nonrepudiation of 468
private keys 467, 470, 483, 484, 485, 500, 501
public keys 467, 470, 487
services 472
signature verification error codes 497
signing 483
signing and encrypting 489
testing for encryption 500
testing for signature 496, 498
trusted CAs 483, 484, 497
verifying signature of 496, 497, 498
Sample View tab 403
SAP BC Developer
main window 28
online help 51
starting 26
supported data types 578
SAP BC Server
access requirements for 26
closing a session 47
connecting to 27, 47
disconnecting from 47
logging on to 27
notification of shutdown 48
opening a session 47
performing ACL checking 93
supported data types 578
SAP BC Type Library 321, 324, 327
Save Pipeline from Server command 278
Save Pipeline Locally command 276
Save Pipeline to Server command 276
savePipelineToFile service 277, 278
saving
input values to a file 251
IS elements 40
test results 275
from Results tab 276
using savePipelineToFile service 277
your work 40
scalar variables
default behavior for mapping 580
description of 580
mapping to or from array variables 580
representation in XML Schemas 349
scale constraining facet 623
Schema Browser, description of 191
Schema Details area, description of 194
schema processor, description of 195
schema services
for node validation 229
for pipeline validation 222
for record validation 229
for XML validation 219
Schema tab 190
scope
of a query 414
property 116
Secure Multipurpose Internet Mail Extensions. See S/MIME
security
assigning ACLs to folders 95
assigning ACLs to services 95
inheriting ACLs 94
Internal ACL 95
managing for services 93
removing ACLs from folders 97
removing ACLs from services 97
Send XML File command 258
sending XML documents
to a service 372
via email 378, 387
via FTP 376
via HTTP 375
sequence content model 193
SEQUENCE step
as target for a BRANCH 115
definition of 73, 568
setting exit condition 125
using in a flow 125
server. See SAP BC Server
server.log file 281
contents of 280
dumping pipeline to 283
overview 280
Service Browser
described 29
icons 29
service element, in WSDL document 337, 347
Service In 151
Service Out 152
service property
definition of 107
editing 107
for transformers 180
renaming transformers with 187
services
assigning ACLs 95
assigning universal names 98
as-user property 97
caching to improve Developer performance 50
creating event filters for 531
creating flow 76
creating with Visual Basic 310, 312
enforcing ACL checks internally 95
externally invoked 93
finding dependents 242
finding invoked 110
flow control 71
generating WSDL documents for 340
inheriting ACLs 94
internally invoked 93
invoking built-in 108
invoking on another server 108
invoking on remote servers 108
provided by SAP BC Server 108
removing ACLs 97
resolving service and record relationships 236
samples in WmSamples 54
security settings 93
stages of creating 75
testing and debugging 248
transformers 177
validating input/output
via Input/Output tab 227
via INVOKE step 228
session
closing on SAP BC Server 47
opening on SAP BC Server 47
restoring 48
session end events
building handlers for 548
default handler 548
description of 547
session.log file 548
session events
See also session end events, session expire events, session start events
description of 524, 546
uses of 547
session expire events
building handlers for 548
default handler 549
description of 547
session.log file 549
session start events
building handlers for 547
default handler 547
description of 547
session.log file 547
session.log file
for session end events 548
for session expire events 549
for session start events 547
Set Breakpoint command 267
SET VALUE modifier 153, 172
copying 174
using in MAP step 173
setting
audit levels 537
breakpoints on flow steps 267
breakpoints on transformers 268
variable values 172
Settings tab 87, 297
for folders 95
for packages 62
SHA-1 message integrity algorithm 468
Shared tab 293
short content constraint 620
shutdown services
definition of 66
guidelines for assigning 66
identifying 67
removing 68
when to use 66
sibling operators 574
signature of a service 288
simple types 193
ing editing in IS schema 198
single view mapping 178
smtp, using to submit XML documents 378
SOAP Arrays
in Web Service Connectors 368
in XML Schema definitions 350
SOAP Message protocol
service requirements for default processor 353
specifying for WSDL documents 352
SOAP processor, default processor requirements 353
SOAP RPC protocol
specifying for WSDL documents 341
source code
compiling Visual Basic 313
compiling with C/C++ make file 310
compiling with jcode 302, 304
creating automatically for a Java service 296
creating from namespace information 303
creating in your own Java IDE 299
creating Java 297
location of 299, 312
tagging for jcode 301
writing in C/C++ 309
writing in Developer 292
Source field on Shared tab 294
Source tab 292
specifications
assigning to a service 208
benefits of 206
creating 206
definition of 206
finding dependents 242
for WebTap services 511
SQL
creating services from 436
specifying dynamic SQL 436
testing 443
ST#.xsd, about 345
stack overflow, during validation 234
starting Developer 26
startTx method 452
startup services
definition of 65
effect on package dependencies 65, 68
guidelines for assigning 66
identifying 67
removing 68
when to use 65
stat events
building handlers for 549
default handler 550
description of 524, 549
stats.log file 550
uses of 549
stats.log file, for stat events 550
status events. See port status events
Step command 260, 264, 265
Step Into command 260, 264, 265
Step Out command 264, 265
stepping through a flow 260, 264
steps. See flow steps
string content constraint 620
String data type
definition of 33, 578
representation in XML Schemas 349, 351
String List data type
converting to Record List 164
defined 578
definition of 33
representation in XML Schemas 349, 351
String Table data type 33, 578
representation as SOAP Array 350
representation in XML Schemas 350, 351
strings
copying by reference 160
transformation services 179
structural constraints 190
  description of 190, 211
  for validation 211
  for XML validation 190
structural transformations
  definition of 150
  examples 164
  mapping to resolve 163
Structured Query Language. See SQL
subfolders
  assigning ACLs 95
  inheriting ACLs 94
  removing ACLs 97
submitTx 457
subordinate flow steps. See children of a flow step
subscribing to events 526
  overview 526
super class, defining in IDE 294
suspending event subscriptions 532
switch property 111, 117
switch value, branching on 110
switching Flow Editor views 140
synchronous transaction
  example of (in flow) 459
  example of (in Java) 454
syntax for conditional expressions 594
T
tables
  creating queries for 414
  support of 579
tabs
  Document View tab 403
  Flow tab 102
  general layout 28
  General tab 171
  Indexing tab 167
  Input/Output tab 84
  Pipeline tab 150
  Properties tab 105
  Rules tab 510
  Sample View tab 403
  Schema tab 190
selecting 35
Settings tab for packages 62
Settings tab for services 87, 297
Shared tab 293
Source tab 292
Variables tab 402
tagging source code for jcode 301
TContext 450
asynchronous transaction 457
synchronous transaction 454
templates
  See also output templates
  assigning to a service 91
  definition of 91
test results
  changing pipeline values 274
  copying 255
  loading 275
  saving 275
  viewing 253
testing services
  entering test input 250
  from a browser 257
  from Developer 249
  inspecting the pipeline 253
  loading input values from a file 252
  loading the pipeline 275
  overview 248
  resolving service and record relationships 236
  run-time exceptions 255
  saving input to a file 251
  saving the pipeline 275
  setting breakpoints 266
  testing in debug mode 259
  viewing results 253
  viewing the call stack 256
  viewing the pipeline 257
  XML document as input 258
threshold value, for maxOccurs 234
time content constraint 621
time conversion services 179
timeDuration content constraint 615
timeDuration content constraint. See duration content constraint
timedInstant content constraint. See dateTime content constraint
timeout property 117, 121, 122, 129
time-to-live period. See TTL
Toggle breakpoint command. See Set Breakpoint command
token content constraint 621
totalDigits constraining facet 215, 624
Trace command 260, 262
Trace Into command 260, 262, 263, 264
and breakpoints 267
Trace to Here command 260, 262, 263
tracePipeline service 283
tracing a flow 260, 262
tracing into a child flow 263
transaction events
  See also Tx End events, Tx Start events
compared to guaranteed delivery events 542
description of 525, 550
txin.log file 552
uses of 551
when generated 542
transaction IDs 452
Transformer not found message 187
transformers
  adding 179
  array variables 183
clearing breakpoints 268
collapsing 185
considerations 179
copying 185
debugging 266
description of 177
dimensionality mismatch 183
disabling 271
enabling 271
expanding 185
inserting 179
mapping 181
mapping array variables 183
movement in Pipeline Editor 182
output variables 181
properties 180
re-enabling 271
renaming 187
service property 180
services provided by SAP BC 179
setting breakpoints 268
stepping in/out of child flows 266
validate-in property 180
validate-out property 180
validating input and output 184
zooming in on 185
trusted CAs 483, 484, 497
TTL
definition of 450
specifying 452
Tx End events
  building handlers for 551
default handler 552
description of 551
uses of 551
when generated 542
Tx Start events
  building handlers for 551
default handler 552
description of 551
uses of 551
when generated 542
txin.log file, for transaction events 552
txout.log file, for guaranteed delivery events 544
type definitions
  anonymous 193
  complex types 193
  simple types 193
types element, in WSDL document 337
typographical conventions 20, 21
U
universal names
  assigning to services 98
  local portion of name 98
  namespace portion of name 98
  removing from a service 99
  used in WSDL documents 345
unresolved links in records 352
unsignedByte content constraint 621
Index

unsignedInt content constraint  621
unsignedLong content constraint  622
unSignedShort content constraint  622
untrusted certificate error  497
update mode (jcode)  304
uriReference. See anyURI content constraint
URL
invoking services from  330, 331
URL encoding restrictions, for WSDL documents  359
user account on SAP BC Server  26

V
validate-in property  109, 180
validate-out property  109, 180
validatePipeline service  222
validating
See also validation
input/output
for transformers  184
via Input/Output tab  227
via INVOKE step  228
nodes  229
pipeline  222
pipeline via built-in services  222
pipeline via Java service  224
records  229
service input/output  226, 227, 228
XML documents  218, 219
validation
See also validating
allowing undeclared variables  213
applying constraints  211
benefits of  210
blueprints for  210
constraining facets  623
constraints, description of  211
content constraints  612
description of  210
ersors  233, 628
exceptions  233, 639
input/output  226
overview  210
requiring variable existence  212
running out of memory  234
service input and output  226
stack overflow  234
types of  210
viewing constraints  216
XML validation  218
validation errors and error codes  628
validation service
pub.schema:validate  219
validation services
pub.schema:validate  219, 229
pub.schema:validatePipeline  222
value transformations, definition of  150
Values objects  290
support for  580
variables
adding to the pipeline  176
addressing in the pipeline  601
applying constraints  211
applying content constraints  612
applying content types  213
checking existence of  596
copying by reference  160
copying by value  160
copying values in pipeline  174
declaring for a service  81, 82, 84
declaring in a record  200
deleting links  169
initializing values  173
linked by blue line  167
mapping  155
mapping conditionally  169
optional existence for validation  212
representation in XML Schemas  348
requiring existence for validation  212
setting properties  205
substitutions  173
using as Set Value  173
viewing applied constraints  216
Variables tab  402
verifying a digital signature  497, 498
version numbers, assigning to packages  61
View as HTML command  99, 154, 203
viewing
assigned value of a variable  172
breakpoints list  269
documentation in PDF format  23
test results  253
viewing this document in PDF format  23
views of Flow Editor, switching between  140
Visual Basic clients
  creating  324
  for database services  445
Visual Basic services
  accessing databases through  442
  creating  310, 312
  early binding  313, 315
  invoking
    early binding service  315
    late binding service  313
  late binding  313
  win32.COM:invoke  313, 315
  win32.COM:invokeLate  313

W
warnings
  flow service generation  643
  IS schema generation  643
  record generation  643
  when generating Web Service Connectors  648
  watt.server.auditLog property  538
  watt.server.stats.pollTime property  545
Web Service Connector
  creating  366
  definition of  366
  example  368
  generation errors and warnings  648
  supporting IS elements  367
Web Services
  and WSDL documents  336
  description  336
Web Services Description Language. See WSDL.
webMethods Query Language
  definition of  410
  object properties  575
  object references  572
overview  572
property masking  576
sibling operators  574
wildcard characters  573
WebTap
  after services  511
  authPwdr  513
  authUrl  513
  authUser  513
  before services  511
  components  506
  control parameters  512
  creating a custom service  518
  creating a standard service  508, 509
  creating rules  510
  custom framework  508, 518
  debugging  516
  definition of  506
  invoking from a browser  520
  mapImages  513
  mapJavascript  513
  mapPage  513
  mapping https URLs  515
  matchSSL  515
  node  515
  reqUrl  512
  reqVals  512
  rerouting example  516
  rspTemplate  512
  rspTxt  513
  rspUrl  512
  rspVals  512
  rules engine  508
  service overview  507
  showApplet  514
  showEmbed  514
  showObject  515
  specification  511
  standard framework  508
  tracing a service  516
  WebControl  512
  WebTapData  512
  whiteSpace constraining facet  216, 624
WIDL
  using in 3.x 382
  using in 4.x 71
wildcard characters, for WQL 573
wildcard symbols
  for child elements 424
  for masking 417
  for pattern matching 414
win32.COM.dispatch:createObject 311
win32.COM:invoke 313, 315
win32.COM:invokeLate 313
windows
  layout 28
  zooming 35
WmDB package 54
WmPartners package 54
WmPublic package 54
WmSamples package 54
WmVBDemo.vpd 312
WmWin32 package 55
WQL 383, 401
  See also webMethods Query Language
WSDL documents 353
  about the generated file 345
  about the generated XML Schemas 348
  address element 348
  description of 336
  elements 336
  errors and warnings 648
  example of 338
  generating
    describing signature with SOAP MSG protocol 353
    files produced 344
    HTTP GET protocol 359
    HTTP POST protocol 359
    service requirements 360
    SOAP Message protocol 352
    SOAP RPC protocol 341
    Web Service Connectors from 366
  generating, process overview 340
  input signature
    identically named variables 354, 361
  specifying 353, 360
  namespaces 339
  naming of elements 346
  output messages 347
  output signature
    identically named variables 354, 361
    specifying 353, 360
    relative paths 345
    universal names 345
    using to create Web Service Connectors 366
    using with default SOAP processor 353
Web Services 336
WSDL generator, definition of 340

X
  X.509 format 472, 483, 487
XML documents
  and $xmldata 373
  converting to Values objects 372
  creating IS schemas from 195
  creating records from 200
  extracting information from 400
  passing to a service 372
  posting via HTTP 375
  retrieving from Web 383
  testing services with 258
  validating 219
XML Query Language
  definition of 410
  methods 419
XML Schema definitions
  creating IS schemas from 195
  creating records from 200
  for WSDL documents 348
  import mechanism 197
  include mechanism 197
  IS schema generation warnings 643
  redefine mechanism 197
  referenced by other XML Schemas 197
XML Schemas
  representation of IS data types 348
XML validation
  content constraints 190
creating IS schemas  195  
description of  218  
errors  628  
exceptions  639  
IS schemas  
  creating  195  
  overview  190  
performing  219  
structural constraints  190  
XQL  383, 401  
  See also XML Query Language  
Y  
year content constraint. See gYear content constraint  
Z  
zones. See drop zones  
zooming  
  in a window  35  
  in on transformers  185