IBM

Developing SIP and IP Multimedia Subsystem (IMS) Applications

- Hands-on introduction to toolkits and development environments
- Learn to develop converged and composite services
- Programming guidelines and working examples

Edward Oguejiofor
Philippe Bazot
Bruno Georges
Rebecca Huber
Callum Jackson
Jochen Kappel
Cameron Martin
Bala S. Subramanian
Abhijit Sur

Redbooks

ibm.com/redbooks

International Technical Support Organization

Developing SIP and IP Multimedia Subsystem (IMS) Applications

February 2007

SG24-7255-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

First Edition (February 2007)

This edition applies to IBM WebSphere Application Server Network Deployment, Version 6.1, IBM WebSphere IP Multimedia Subsystem Connector V6.1, IBM WebSphere Presence Server V6.1, IBM WebSphere Telecom Web Services Server V6.1 and IBM WebSphere® Integration Developer Version 6.0.

Contents

Notices . . . xi
Trademarks . . . xii

Preface . . . xiii
The team that wrote this redbook . . . xiv
Become a published author . . . xvii
Comments welcome . . . xviii

Part 1. Introduction to SIP and IMS . . . 1

Chapter 1. Introduction to Session Initiation Protocol (SIP) . . . 3
1.1 SIP overview . . . 4
1.2 SIP architectural components . . . 7
1.3 SIP messages . . . 9
1.3.1 SIP requests . . . 11
1.3.2 SIP responses . . . 13
1.3.3 SIP transactions . . . 15
1.3.4 SIP dialogs . . . 16
1.3.5 A sample SIP call flow . . . 17
1.4 SIP Java development . . . 20
1.4.1 JAIN SIP API . . . 20
1.4.2 SIP Servlet API . . . 23
1.5 Examples of SIP application . . . 26

Chapter 2. Introduction to IP Multimedia Subsystem . . . 27
2.1 IMS overview . . . 28
2.1.1 IMS vision and history . . . 28
2.2 Elements of IMS architecture . . . 30
2.2.1 Functional components . . . 31
2.2.2 Reference points . . . 33
2.2.3 Protocols . . . 35
2.2.4 Functional planes . . . 36
2.3 Services in IMS . . . 38
2.3.1 Service architecture . . . 38

Part 2. Application development technologies . . . 41

Chapter 3. Introduction to IBM SIP and IMS service creation . . . 43
3.1 Overview . . . 44

3.2 IBM Unified Service Creation Environment 45
3.3 Types of SIP and IMS applications 49
3.4 The SIP and IMS service creation environment 51
3.4.1 IBM WebSphere Application Server Toolkit 53
3.4.2 IBM IMS Enablement Toolkit. 55
3.4.3 IBM Telecom Web Services Toolkit 57
3.4.4 IMS Enablement Toolkit 58
3.5 The service execution environment. 60

Chapter 4. IBM WebSphere Application Server Toolkit 63
4.1 AST overview 64
4.2 Developing SIP servlet application 66
4.2.1 SIP only applications. 67
4.2.2 Converged SIP/HTTP applications 69
4.2.3 SIP servlet deployment 71
4.2.4 Sample SIP services 82
4.2.5 Hardware and software requirements 88

Chapter 5. IBM IMS Enablement Toolkit 89
5.1 IMS Enablement Toolkit overview 90
5.2 Developing IMS foundation applications 90
5.2.1 Diameter client application 97
5.2.2 Presence Server components. 100
5.2.3 Parlay X Web Services 101
5.3 Sample IMS foundation applications. 101
5.3.1 ISC Interface sample. 101
5.3.2 Diameter client samples 110

Chapter 6. IBM WebSphere Integration Developer. 119
6.1 Overview 120
6.2 Working with IBM WebSphere Integration Developer 120
6.2.1 Key concepts. 120
6.2.2 Modules. 122
6.3 Components 123
6.3.1 Business Integration perspective and views 126
6.3.2 Adding custom logic to BPEL processes 128
6.4 IMS service components 130
6.4.1 Assembling components 133
6.4.2 Component tests 134
6.5 Technical information 141
6.5.1 Packaging 141
6.5.2 Supported platforms 142

Chapter 7. IBM Telecom Web Services Server Toolkit. 145

7.1 Introduction 146
7.1.1 Mediation services 147
7.1.2 TWSS Mediation primitives 148
7.1.3 TWSS default message flow 149
7.2 The IBM Telecom Web Services Server Toolkit 150
7.2.1 Importing TWSS mediation flows 151
7.2.2 Working with TWSS mediation flows 154

Chapter 8. Introduction to the IBM service execution environment 159
8.1 Overview of the IBM IMS solution 160
8.1.1 The service execution environment 162
8.2 The IBM WebSphere Application Server. 165
8.2.1 WebSphere Application Server SIP support 167
8.3 WebSphere IMS Connector 173
8.3.1 ISC interface 174
8.3.2 Diameter services 178
8.3.3 IBM WebSphere Diameter Enabler. 180
8.4 WebSphere Presence Server 182
8.4.1 IBM WebSphere Presence Server Component 182
8.4.2 The Presence Management enabler. 184
8.5 IBM WebSphere Group List Server. 185
8.5.1 The role of Group List Management 187
8.5.2 XDM/XCAP Interface 190
8.6 Telecom Web Services Server 193
8.6.1 Telecom Web Services Access Gateway 194
8.7 Telecom Web Services Server service implementations 196
8.7.1 Common components. 198
8.7.2 Service Policy Manager 199
8.8 WebSphere Enterprise Service Bus 199
8.9 WebSphere Process Server 201

Part 3. SIP applications 203

Chapter 9. Developing SIP applications 205
9.1 Overview of SIP applications. 206
9.1.1 SIP Servlet container 206
9.2 SIP Servlet 207
9.2.1 Differences between SIP and HTTP Servlet. 207
9.2.2 Converged servlet. 209
9.3 Elements of SIP applications. 209
9.3.1 Receiving requests 209
9.3.2 Parsing messages. 210
9.3.3 Creating responses. 212
9.3.4 Creating requests 213

9.3.5 Receiving responses 216
9.3.6 Proxies 217
9.3.7 Mapping requests to servlets 219
9.3.8 Sessions 221
9.3.9 Listeners and events 223
9.3.10 Timers 225
9.3.11 Security 226
9.3.12 Converged servlet 227
9.4 Best practices 228
9.4.1 Application layering 229
9.4.2 Message processing 230
9.4.3 Implement specification design requirements 230
9.4.4 Runtime development considerations 231

Chapter 10. Sample SIP applications 233
10.1 Application overview 234
10.2 Registrar and proxy application 234
10.2.1 The scenario 234
10.3 Creating the SIP application project 235
10.3.1 Developing the SIP Servlets 238
10.3.2 Configure User Agents 252
10.3.3 Testing the Registrar and proxy application 253
10.4 Third Party Call Control application 256
10.4.1 Overview 257
10.4.2 Develop using the Application Server Toolkit 258
10.4.3 Compose the Application 278
10.4.4 Deploy the converged SIP/J2EE application 280
10.4.5 Testing the Third Party Call Control application 286
10.4.6 Debug and trace the application 289

Part 4. Developing IMS applications 299

Chapter 11. Designing IMS services 301
11.1 Overview of IMS composite services 302
11.1.1 Composite services architecture 302
11.2 Composite services choreography 303
11.2.1 Composite services orchestration 305
11.3 Designing composite services 305
11.3.1 Design process 305
11.3.2 SIP Servlets as Web Services 307
11.3.3 Deciding when to use BPEL 311
11.3.4 Choosing ESB Software 312
11.4 Sample application design 312
11.4.1 Objectives of the sample application 312

11.4.2 The business scenario . . . 313
11.4.3 The use case model . . . 314
11.4.4 The component model . . . 317
11.4.5 Component flow . . . 322
11.5 SIP Servlet design . . . 328
11.5.1 BPEL design . . . 336

Chapter 12. Implementing the IMS sample service . . . 341
12.1 Implementation overview . . . 342
12.2 SIP Servlet development . . . 342
12.2.1 Create a new SIP project . . . 342
12.2.2 Create a new SIP Servlet . . . 344
12.2.3 Complete the SIP Servlet code . . . 352
12.2.4 Export the application for deployment . . . 360
12.3 BPEL development . . . 361
12.3.1 Create a new business integration module . . . 361
12.3.2 Create the business object . . . 364
12.3.3 Create the interface for the BPEL process . . . 368
12.3.4 Import the WSDL files . . . 373
12.3.5 Create the business process . . . 383
12.3.6 Add partner references . . . 387
12.3.7 Add process logic . . . 390
12.3.8 Assemble the FindHelp module . . . 416
12.4 Export the FindHelp WSDL files . . . 427
12.4.1 Unit test the FindHelp module . . . 430
12.5 The location simulator . . . 435

Chapter 13. Sample IMS application test environment . . . 439
13.1 Overview of the test environment . . . 440
13.2 Setting up the test environment . . . 445
13.2.1 Group List Server setup . . . 445
13.2.2 Location server setup . . . 448
13.2.3 Application deployment . . . 450
13.2.4 Device client setup . . . 459
13.2.5 Installing the IBM Diameter CCF Simulator . . . 465
13.3 Executing the test scenarios . . . 466
13.3.1 Use Case 1: Administrator adds service topic . . . 467
13.3.2 Use Case 2: Publish Technician Status . . . 469
13.3.3 Use Case 3: Caller requests FindHelp Service . . . 473
13.4 Problem determination and resolution . . . 480
13.5 Step-by-step tracing . . . 480
13.5.1 Enable SIP debug tracing on the Linux test server . . . 486
13.5.2 Tracing SIP messages using Ethereal . . . 488

13.6 Log files 493

Part 5. Appendixes 497

Appendix A. Installing the application development environment 499
A.1 Installing the SIP AST 500
A.1.1 Starting the SIP AST 502
A.2 SIP device client installation 502
A.2.1 SipXphone 503
A.2.2 X-Lite 505
A.2.3 SJPhone 507
A.3 Installing the IMS Enablement Toolkit 508
A.3.1 Verify the installation of the IMS Enablement Toolkit 513
A.4 Installing WebSphere Integration Developer 515
A.4.1 Update WebSphere Integration Developer 521
A.4.2 Apply required fixes 522
A.5 Installing the Telecom Web Services Server plug-in 524
A.5.1 Extract the ESB mediation flows and import them into WID 527

Appendix B. Installing the sample application test environment 531
B.1 IBM WebSphere Application Server 6.1 532
B.2 IBM WebSphere Telecom Web Services Server 540
B.2.1 Create the WebSphere Application Server profile 541
B.2.2 Install base binaries 544
B.2.3 Configure DB2 546
B.2.4 Configure Service Integration Bus 554
B.2.5 Configure JDBC 561
B.2.6 Tune the Application Server 567
B.2.7 Deploy TWSS Applications 570
B.2.8 Verify the installation 582
B.2.9 Troubleshoot the installation 583
B.3 IBM WebSphere Group List Server component 586
B.3.1 Install base binaries 587
B.3.2 Create WebSphere Application Server profile 587
B.3.3 Configure DB2 589
B.3.4 Configure the LDAP directory 591
B.3.5 Configure users and groups 592
B.3.6 Configure JDBC and data sources 594
B.3.7 Tune the Application Server 599
B.3.8 Deploy GLS application 603
B.3.9 Install the Self Care portlet 606
B.3.10 Install the command line interface 609
B.3.11 Administration 609
B.4 IBM WebSphere Presence Server component 611

B.4.1 Install base binaries 611
B.4.2 Create WebSphere Application Server profile 611
B.4.3 Configure DB2 612
B.5 Create the Service Integration Bus and bus members 615
B.5.1 Configure JDBC and data source 618
B.6 Deploy PS application 621
B.7 IBM WebSphere Diameter Enabler component 625
B.7.1 Install base binaries 626
B.7.2 Create WebSphere Application Server profile 626
B.7.3 Deploy the Diameter Enabler application on WebSphere Application Server 627
B.7.4 Deploy Diameter Rf Web Services 634

Appendix C. Additional material 637
Locating the Web material 637
Using the Web material 637
System requirements for downloading the Web material 638
How to use the Web material 638

Abbreviations and acronyms 639

Related publications 643
IBM Redbooks 643
Other publications 644
Online resources 644
How to get IBM Redbooks 645
Help from IBM 645

Index 647

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

ibm.com®	DB2 Universal Database™	Rational®
pSeries®	DB2®	Redbooks™
xSeries®	IBM®	Redbooks (logo) ™
AIX®	IMS™	Tivoli®
BladeCenter®	Rational Unified Process®	WebSphere®

The following terms are trademarks of other companies:

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates.

Enterprise JavaBeans, EJB, Java, JavaBeans, JavaScript, JavaServer, JavaServer Pages, JAIN, JDBC, JDK, JMX, JSP, JVM, J2EE, J2ME, J2SE, Solaris, Sun, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

DirectX, Expression, Internet Explorer, Microsoft, Windows Server, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Pentium, Xeon, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

The convergence of Internet Protocol (IP) networks is enabling seamless communications that combine data, voice, video and other information streams. The true value of a converged IP network, however, is realized through the converged applications that use the network productively. The key enabler to the development of converged applications is the platform, for designing, developing, testing, and deploying applications that integrate and compose services.

This IBM® Redbook introduces IBM tools for creating converged Session Initiation Protocol (SIP) and IP Multimedia Subsystem (IMS™) applications. It provides programming guidelines and working examples that demonstrate how to use the different development tools. It also provides hints and tips that can get you up to speed quickly for developing converged applications.

The portfolio of products includes the IBM WebSphere® Application Server Network Deployment, IBM WebSphere IP Multimedia Subsystem Connector, IBM WebSphere Presence Server, IBM WebSphere Telecom Web Services Server, and IBM WebSphere Integration Developer.

This book is structured into five parts:

- Part 1, “Introduction to SIP and IMS”, provides an overview of SIP and the different components and protocols that make up SIP. It also presents an overview of IMS, the specifications that define 3G (third generation) architecture for telecommunications services.
- Part 2, “Application development technologies”, provides an overview of the IBM service creation and execution environment. It highlights the features of the different tools that are used for creating SIP, converged SIP and HTTP, IMS foundation and IMS composite applications.
- Part 3, “SIP applications”, provides the programming guide for developing SIP based converged applications. It includes working examples that demonstrate use of IBM tools for application development.
- Part 4, “Developing IMS applications”, provides guidelines for developing applications that use the IBM WebSphere IP Multimedia Subsystem Connector, the IBM WebSphere Presence Server and the IBM WebSphere Telecom Web Services Server. The working examples demonstrate how to create composite services.
- Part 5, “Appendixes”, provides additional information about the installation and configuration of the test environment and references to related

publications and other useful information, including how to obtain the source code for the sample applications.

This redbook is geared toward the diverse set of professionals that have the responsibility for designing and developing SIP and IMS applications. These include IT architects that develop the solution, and IT specialists, application integrators, and developers who design, code, and test converged applications and composite services.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

The IBM redbook team (front row - left to right: Edward Oguejiofor, Rebecca Huber, Callum Jackson, and Bruno Georges; back row - left to right: Jochen Kappel, Philippe Bazot and Cameron Martin). Participating remotely were Bala S. Subramanian and Abhijit Sur, pictured below.

Edward Oguejiofor is a Project Leader at the International Technical Support Organization, Raleigh Center. He has over 20 years experience in distributed and enterprise systems architecture and design. His areas of expertise include Web Services and service-oriented architecture. He also provides technical and thought leadership in emerging technologies and collaborative services. He holds a degree in Computer Science from Imperial College of Science and Technology, University of London.

Philippe Bazot is an IBM Certified IT Specialist and works at the European Business Solutions Center at IBM La Gaude, France. He is currently working for IBM WebSphere Business Partner Enablement and provides EMEA advanced technical support to IBM Business Partners, by assessing and enabling them to use IBM WebSphere IMS products portfolio. He has 23 years of experience in networking software. He is also a Solution Architect, working in various areas, such as Service Provider Development Environment, telco platforms, telephony, networking and Internet. He holds a diploma from Ecole Superieure d'Electricite in France.

Bruno Georges is an IBM Certified IT Specialist and works as a WebSphere Technical Sales Specialist for South-East IOT Telco customers, mainly for Service Delivery Portal and IMS solutions, from the IBM Center in La Gaude, France. He has six years of experience in the wireless and telco industries. He has worked at IBM for 21years. His areas of expertise include application development, Java™ Application Servers, and Portal and their mobile extensions, wireless technologies. He holds a diploma from Ecole Natioanle Superieure des Arts et Metiers and a master’s degree from the University of California (Berkeley).

Rebecca Huber is a Senior IT Architect in IBM Global Business Services, Germany. She has 19 years of IT experience in the fields of application development and project planning, and she has worked the last three years on mobile telecommunications topics such as IMS, next generation HLR, and voice messaging. She is an IBM certified Telecommunications IT Architect. Her areas of expertise include end-to-end systems integration, application architecture and development, and technical team leadership. She holds a degree in Computer Science from Clemson University, South Carolina, USA.

Callum Jackson is an IT Specialist in ISSW, based in Hursley, United Kingdom, and he specializes in Voice and IMS solutions. His engagements primarily regard WebSphere Voice Response, WebSphere Voice Server and WebSphere Application Server, WebSphere Process Server and WebSphere Portal. He has more than five years experience working with Voice and integration with J2EE™ solutions. The past two years or more he has been involved in IMS solutions for various customers.

Jochen Kappel is a Senior IT Architect in the IBM Telecom Solutions Lab in Europe. He joined IBM in 2001, focusing on systems design that uses IBM Service Provider Delivery Environment. He has 20 years of experience in building solutions for wireless and telecommunication operators as well as software development methodologies. He holds a diploma in Electrical Engineering and Telecommunications from the Technical University Darmstadt, Germany.

Cameron Martin is a Consulting IT Specialist working for IBM Software Group in Hursley, United Kingdom. He has over ten years of experience in the IT industry, with six of those years spent advising clients on complex WebSphere Application Server deployments across Europe and Asia Pacific. He is currently specializing in SIP, IMS and, J2EE applications and infrastructures, with a focus in the areas of performance, security and high availability. He holds a bachelor's degree in Business Information Technology from the University of New South Wales, Australia.

Bala S. Subramanian is an IBM certified Senior IT Specialist for Business Applications from Telecom and Media Dept., IBM Services, India. He is a Board Member of Advisory Accreditation for IT Specialist and Asia Pacific IT Specialist Discipline Leader. He has more than six years experience in the IT Industry and specializes in service-oriented architecture (SOA). His expertise includes AIX®, Linux®, Solaris™, WebSphere products and programming in Java, J2EE, Web Services, portals, and XML. He focuses primarily in designing and developing solutions, that use Java, J2EE, Web Services, and portal solutions. He holds five IBM certifications, including: IBM WebSphere Application Server, WebSphere Studio Developer, developing Web Services with WebSphere Studio and DB2® Universal Database™. He is also a Sun™ Certified Java Programmer and SEI PSP Level 5 Engineer from SEI, Carnegie Mellon University, Pittsburgh. Additionally, he holds a degree in Computer Engineering from Madurai Kamaraj University, TamilNadu, India.

Abhijit Sur has over 12 years of telecommunication industry experience spanning several areas of OSS/BSS solutions, such as process improvements, functional analysis, customization, implementation, configuration, and system integration. He has been with IBM since 2002. Currently, as an Advisory Solution Architect for the Communications sector, he is involved in designing service-oriented architecture for third generation cellular operators and cellular carriers, transforming their networks for a converged IMS platform. He holds a master's degree in Statistics from IIT Kanpur, India; and in Telecommunications from the University of Colorado (Boulder). He has published several papers and articles in technical journals and magazines. His research interests include Vertical Handoffs, Spectrum Management and Web Services based service-oriented architecture for next generation networks.

Thanks to the following people for their contributions to this project:

IBM Software Group

- Joan Boone
- Scott Broussard
- Erik J. Burckart
- Girish Dhanakshirur
- Joel Dunn

- Ratna Garimella
- Yun Huang
- Ronnie Jones
- Jennifer King
- Tim Smith
- Ed Spring
- Cristi Nesbitt Ullmann
- Anthony Wrobel

IBM Software Group, WPLC, Haifa

- Danielle Volski
- Dror Yaffe

IBM Sales and Distribution, Communications Sector (Germany)

- Josef Reisinger

International Technical Support Organization, Raleigh Center

- Tamikia Barrow
- Margaret Ticknor
- Jeanne Tucker
- Linda Robinson

International Technical Support Organization, Poughkeepsie Center

- Michael B. Schwartz

International Technical Support Organization, Business Controls

- Erica B. Wazewski

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

`ibm.com/redbooks/residencies.html`

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Part 1

Introduction to SIP and IMS

Part 1 provides an overview of SIP and the different components and protocols that make up SIP. It also presents an overview of IMS, the specifications that define 3G (third generation) architecture for telecommunications services.

1

Introduction to Session Initiation Protocol (SIP)

This chapter provides an overview of Session Initiation Protocol (SIP), the different components that make up SIP, and the programming interface for creating SIP applications that use Java programming language.

This chapter contains the following:

- SIP overview
- SIP architectural components
- SIP messages
- SIP Java development
- Examples of SIP application

1.1 SIP overview

Session Initiation Protocol (SIP) is an application-layer protocol for initiating, modifying, or terminating communication and collaborative sessions over Internet Protocol (IP) networks. A session could be an IP telephony call, a multi-user conference that incorporate voice, video and data, instant messaging chat or multi player online game. SIP can be used to invite participants to a scheduled or already existing session. Participants can be a person, an automated service or a physical device such as a handset. It can also be used to add or remove media to a session.

A bit of history

SIP emerged in the mid-1990s from two proposals, Session Initiation Protocol (SIP) by Mark Handley and Eve Schooler, and the Simple Conference Invitation Protocol (SCIP) by Henning Schulzrinne that were submitted to the Multi-Party, Multimedia Working Group of the Internet Engineering Task Force (IETF). The two proposals were merged to form Session Initiation Protocol and was approved as RFC 2543 by the IETF in March 1999. The final standard was released as a part of RFC 3261 in 2002.

Interest in SIP has continued to grow and separate Work Groups are contributing to the core standard and extensions that leverage SIP. The SIP Working Group (WG) is chartered to maintain and continue the development of SIP as proposed by RFC 3261. The SIP Working Group considers change requirements from other working groups that develop systems based on SIP such as the SIPPING (Session Initiation Protocol Project INvestiGation) working group, which analyzes the application of SIP, the SIMPLE (Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions) working group which is using SIP for messaging and presence, and the XCON working group using SIP for centralized conferencing.

Capabilities

In telecommunication networks, there are two categories of traffic. First of all there is traffic related to control signaling (hereafter referred to as signaling) which is used to establish, manage and terminate communication sessions. And then there is the actual data traffic (for example voice). SIP signaling follows the concept of common channel signaling, whereby the path used for the signaling traffic is independent of the path used for the actual data traffic. Separation of signaling traffic from media makes the session management more efficient, and is also more adaptive to functional changes.

SIP signalling supports the following facets of multimedia session management.

- User location

  Enabling users to access telephony or other application features from remote locations.

- User availability

  Determining the willingness of the called parties to engage in communication sessions.

- User capabilities

  Determining the media and media parameters to be used for communication sessions.

- Session setup

  Establishing the session parameters for point-to-point and multiparty sessions.

- Session management

  Enabling the transfer and termination of sessions, the modification of session parameters, and the invocation of session services.

Working with other protocols

SIP leverages existing Internet application protocols and standards such as the Simple Mail Transfer Protocol (SMTP) and Hypertext Transfer Protocol (HTTP). While HTTP provides integration of content (text, audio, video, links to other Web pages) on Web pages, SIP integrates different media into sessions. SIP adopted the request/response paradigm of HTTP, and many HTTP message headers and codes (for example, error code - 404: Address not found). However there are some key differences between SIP and HTTP. Unlike HTTP, SIP is a peer-to-peer protocol. With HTTP, a Web server doesn't originate requests. whereas any SIP user agent can send a request to initiate or modify a session. Moreover, SIP has the provision of generating multiple responses (and possibly to multiple destinations) from a single request. Another key difference from architecture perspective is that HTTP services are typically hosted on an HTTP server (which generates the response to requests), SIP servers (proxy server, Redirect Server and Registrar discussed in 1.2, “SIP architectural components” on page 7) provide intelligent services routing.

SIP uses Uniform Resource Identifier (URI) to identify participants and resources in communication sessions. SIP URI is in a form similar to the mailto URL. Its general form is:

```
sip:user:password@host:port;uri-parameters?headers
```

The following are examples of SIP URI:

- Calling a PSTN (Public Switched Telephone Network) phone

  `sip:1-112-233-4455@pstnnetwork.com;user=phone;`

  SIP parameters can be used to provide additional information, in this example user=phone is used to indicate that the call is to a telephone number.

- Internet Call

  `sip:123-4567@itso.com;user=phone; context=privnet;`

  Context=privnet, in this case, indicates a private IP network, such as a company's corporate network, and the PSTN.

- Calling a personal computer

  `sip:johndoe@itso.com`

  You can also use a secure link, for example – sips:johndoe@itso.com. SIPS indicates refers to a secure connection in the same way as https.

SIP uses DNS procedures to resolve SIP Uniform Resource Identifiers into the IP address, port, and transport protocol. It also uses DNS to allow servers to send response to backup clients if the primary client has failed.

For end-to-end functionality and services such as voice over IP (VoIP), SIP works with a number of other protocols. Session Description Protocol (SDP) defines the format for describing the characteristics and parameters of multimedia sessions. The Real-time Transport Protocol (RTP) defines a standardized packet format for delivering audio and video over the Internet, and Real-time Transport Control Protocol, (RTCP) provides out-of-band control information about the quality of the RTP data flow.

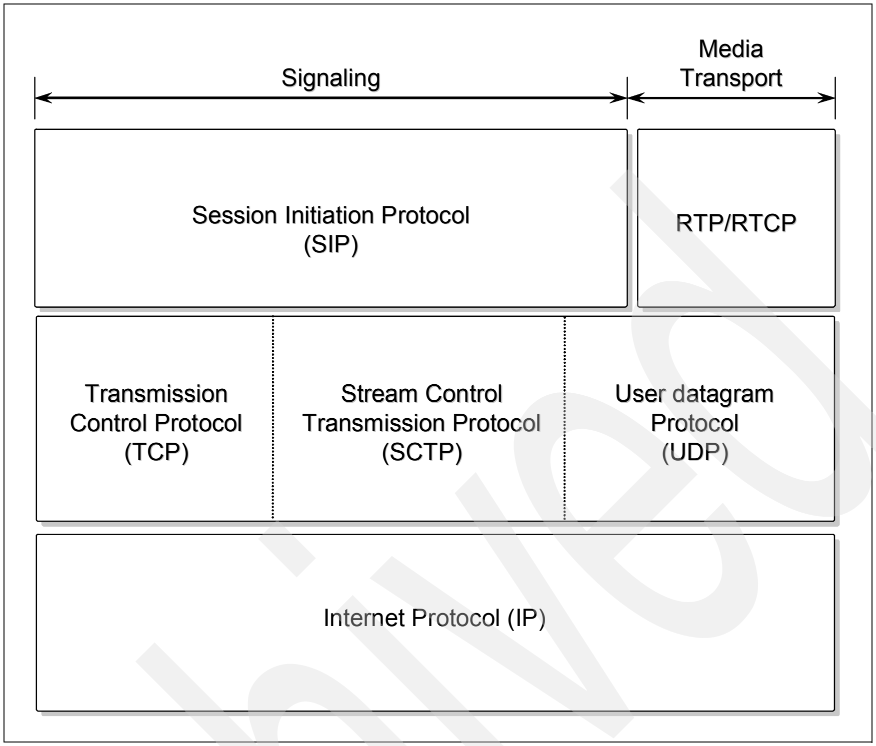


Figure 1-1 The SIP communication stack

1.2 SIP architectural components

In this section, we describe the architectural components of SIP - the user agent and network servers. SIP messaging and the extensions which is introduced in 1.3, “SIP messages” on page 9 show how these components participate in call flow to support SIP functionality.

Broadly speaking SIP networks consist of two basic components - SIP user agent and SIP network server. The SIP user agents are the peer components that initiate and answer calls.

SIP architecture define the following functional elements:

- User Agent

  A SIP User Agent (UA) is an end device which can originate and receive SIP calls. It can be a phone terminal (mobile, PDA or Laptop) or an endpoint such as an answering machine. SIP supports both peer-to-peer and client server architectures. User agents act as peers, they find each other and negotiate session characteristics.

- User Agent Server

  In the client server model, when sending requests or receiving responses, a SIP UA acts as the client, in which case it is referred to as the User Agent Client (UAC). The receiving SIP UA acts as the server (receives and requests and sends responses) and is referred to as the User Agent Server (UAS). UAC and UAS are logical entities that are contained in every SIP User Agent.

- Back-to-Back UA (B2BUA)

  When a SIP entity acts both as a User Agent Client, as well as the User Agent Server it is referred to as the Back-to-Back User Agent (B2BUSA). It generates requests to determine how the incoming request is to be answered.

- Proxy server

  The SIP proxy server is a key component of SIP infrastructure. Its role as an edge routing server is similar to that of a Web proxy server. It provides routing capabilities and supports functions such as authentication, accounting, registration and security. The SIP proxy server is the first entity that receives all outgoing requests from a SIP UA, it routes the request traversing intermediate servers until it locates the server closest to the destination SIP UA, which forwards the request to the called SIP UA. In the most common scenarios there are usually two SIP proxy servers - one at the caller end and one at the callee end.

  Proxy Servers can be configured to be transaction stateful or stateless. Stateless proxy servers receive incoming requests and simply pass on responses without retaining any information about the transaction. On the other hand, stateful proxy servers retain information about all incoming requests, the server's responses and outgoing messages from the server. A SIP infrastructure can contain a combination of stateful and stateless proxy servers. The stateful servers can be configured closer to the SIP user agents to collect billing and other user relevant data, whereas the stateless proxy servers form the backbone of the network.

  SIP proxy servers can perform 'forking' process, where it sends out SIP INVITE to multiple devices at once. In the case where a user is registered at multiple locations, a Forking Proxy Server would send an incoming SIP INVITE message (for a session) to each registered location. When the user responds from one of the locations (upon receiving a SIP OK message from that location) the proxy server sends a SIP CANCEL message to the other locations. In order to perform forking process, the proxy server needs to be configured to be transaction stateful.

- Registrar

  The Registrar is a repository of user agents location information. The registrar accepts registration requests from user agents and places the information (the sip address and associated IP address) in location database. A SIP Register message will tell the Registrar (and the network) at which address (or multiple addresses) the user will be available henceforth (say office phone during the day). Once the location or device changes the user agent has to send another SIP Register message to the Registrar. SIP proxy servers (and redirect servers) make use of the location information stored in the repository to obtain the callee user agent location(s).

- Redirect Server

  Redirect Servers respond to SIP request with an address where the SIP message should be redirected. It maps a destination address (in the SIP message) to one or more addresses and returns the new address list to the originator of the SIP request. The location of the intended recipient is retrieved from the location database maintained by the SIP Registrar. Redirection is used for Call Forwarding and it also helps to reduce the processing load on proxy servers by pushing the processing back to the requesting clients.

1.3 SIP messages

SIP signaling - the setting up, modification and termination of communication and collaboration sessions - is realized through the exchange of messages. There are two types of messages: requests and responses. Requests are sent to initiate some action and responses are sent as replies to requests acknowledging receipt of requests and indicating the processing status.

Requests and responses share a common message format which consist of a start-line, one or more header fields, an empty line indicating the end of the header fields, and an optional message-body. The SIP message structure is illustrated in Figure 1-2 on page 10.

Figure 1-2 Structure of SIP messages

> **Note:** The start-line, message-header line, and the empty line are terminated by a carriage-return line-feed sequence (CRLF). The message body is optional, however the empty line is present even if the message-body is not.

The start-line in SIP messages can be either a request or a status line. Request messages use the request line to set the type of request. Response messages indicate whether the processing of a request is successful or not in the status line.

SIP message headers consist of fields with name value pairs. Where some fields are optional such as content type and length, some fields are mandatory for every SIP message. Table 1-1 on page 11 list the mandatory header fields for SIP messages.

Table 1-1 SIP message mandatory header fields

Field name	Description
To	The request destination’s SIP address
From	Indicates the originator of the request
CSeq	The command sequence that ensures messages are dealt with, in the order they were generated.
Call-ID	A randomly generated string that uniquely identify SIP sessions. SIP proxy servers use Call-ID to identify messages belonging to a SIP session.
Via	Contains information about SIP devices a message has passed through as it moves between caller and callee. The Via field is also used to route responses in the reverse direction.
Contact	Contains the actual location of the callee, which might be different from the address of the originator in the From header

1.3.1 SIP requests

SIP requests have a Request-line for their start-line. The format of a Request-line is illustrated in Figure 1-3. It consists of three fields that are separated by a single space (SP) character.

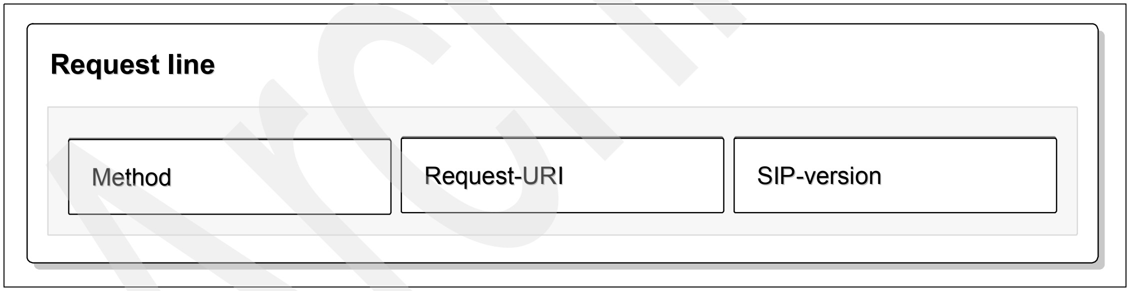


Figure 1-3 Format of a request message start-line

- Method

  This field indicates the method to be performed. RFC 3261 identified six methods: REGISTER, INVITE, ACK, CANCEL, BYE and OPTIONS. SIP extensions documented in other RFCs have defined additional methods. Table 1-2 on page 12presents a list of request methods and a brief description of what each method does.

Table 1-2 List of SIP request methods

Method	Description
REGISTER	This method is used to provides the Registrar with information specifying the UA's location and available for incoming SIP requests. When the user agent's location changes, another REGISTER message is sent to update the Registrar's database.
INVITE	This method is used to initiate a communication session between two UA peers. sent message is sent by a user to initiate a session with another peer user. INVITE can also be used to initiate a multi party call.
ACK	This method is used for acknowledgement. It indicates that the final response has been received.
CANCEL	This method is used to terminate pending requests. A calling party can cancel an INVITE message before it receives the final response.
OPTIONS	This method is used to query a server on its capabilities. For example, it can be used to query if a to-be-called party can support a particular type of media.
BYE	This method is used to indicate the termination of a session.
INFO	This method (an extension - RFC 2976) is used to communicate additional information about an active session. For example, it can be used to inform a user agent when available pre-paid balance is approaching zero.
REFER	This method (an extension - RFC 3515) is used to provide the recipient with a third party's contact information. This method can be used for call transfers.
UPDATE	This new method (RFC 3311) is used to change session information (such as a change in codec) before a final response to a SIP INVITE message has been received. Typically, UPDATE messages contain an SDP body that lists the modified session parameters.
SUBSCRIBE and NOTIFY	These are two new SIP methods (RFC 3265) that are used in conjunction with each other for SIP based event notification. A user can subscribe to events such as the Presence information of another user. Subscribe Method s sent to the SIP Presence Server, so when the second user becomes available, the Presence Server sends a NOTIFY.

- Request-URI

  The request URI field holds a SIP or SIPS URI. It is used to indicate the user or service to which the request is addressed.

- SIP-version

  The SIP version field identifies the version of SIP protocol that is in use.

> **Note:** The request-line ends with a carriage-return line-feed sequence (CRLF). No CR or LF are allowed except in the end of line CRLF sequence. Also, no linear whitewashes is allowed in any of the elements.

1.3.2 SIP responses

The receipt of a request by a user agent or a proxy server triggers a response which is sent as a SIP response message.

> **Note:** ACK requests do not trigger response.

SIP response messages are distinguished by the fact that they have Status-line in their start-line. The Status-line consist of three fields SIP-version, Status-code and Reason-phrase which are separated by a single space (SP) character. The format of a Status-line is illustrated in Figure 1-3 on page 11.

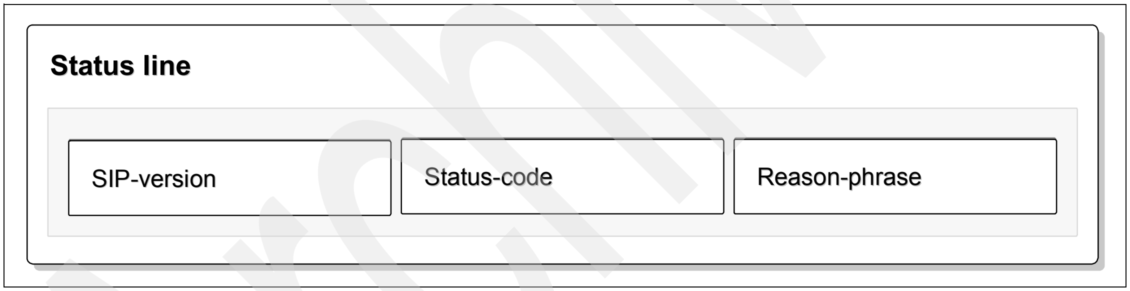


Figure 1-4 Format of a response message start-line

- SIP-version

  The SIP version field identifies the version of SIP protocol that is in use.

- Status-code

  The status-code is a three digit code which represent the outcome of request processing. The range of values is between 100 and 699. The first digit indicates the class of the response (SIP/2.0 allows for 6 possible classes). Table 1-3 provides a brief overview of the response classes.

Table 1-3 Response message status-code classes

Code range	Description
1xx	► Provisional/Informative response Provisional response indicate that the associated request was received and being processed. Upon receipt of a provisional response, the request sender should stop retransmitting the request. For example, a proxy server can send a response message with a Status-code of 100 upon receipt of an INVITE request. Provisional responses need not be acknowledged with an ACK.
2xx	► Success Success responses with Status-codes in the range from 200 to 299 indicate that the request was received, understood and successfully processed. For example, a 200 OK response is sent to a User Agent Client when an INVITE request is successfully processed.
3xx	► Redirection When further action such as a different location is needed to complete a request, redirection responses are used to provide the new location or an alternative service that would satisfy the request. Redirection responses are usually sent by redirect servers. When a redirect server receives a request, it maps the destination address in the request message to one or more addresses retrieved from the Registrar location database and returns the new address list to the originator of the request. The originator is then supposed to re-send the request to the new location.
4xx	► Client error Client error response Status-codes are sent when requests cannot be processed. The request failure could be because of bad syntax in the request message or simply because the request cannot be fulfilled by the responding server.
5xx	► Server error Server error response Status-codes are sent in cases where the request is valid but the server s unable to fulfill the request. Server internal error (500) and Not implemented (501) are two examples of Server error response Status-codes.
6xx	► Global failure When a request cannot be fulfilled by any server, the Global failure response Status-codes are returned. A User Agent Server can return a global failure response with Status-code 603 to Decline a request to participate in a session.

- Reason-phrase

  This is a short textual description of the Status-code. Where the Status-code is intended for machine processing, the reason-phrase is a human-readable message that is rendered to the user by the user agent.

1.3.3 SIP transactions

The set of messages exchanges by SIP components starting with the initial request and all responses related to the request (including zero or more provisional and any final responses) are considered a SIP transaction. If the request is an INVITE, then the ACK is considered part of the transaction only if the final response is not a 2xx response (that is a negative outcome).

Processing SIP transactions require maintenance of state information. Consequently, SIP components that associate requests and responses to transactions are considered stateful. Stateful SIP components extract unique transaction identifiers from messages and are able to update the state information for the transaction.

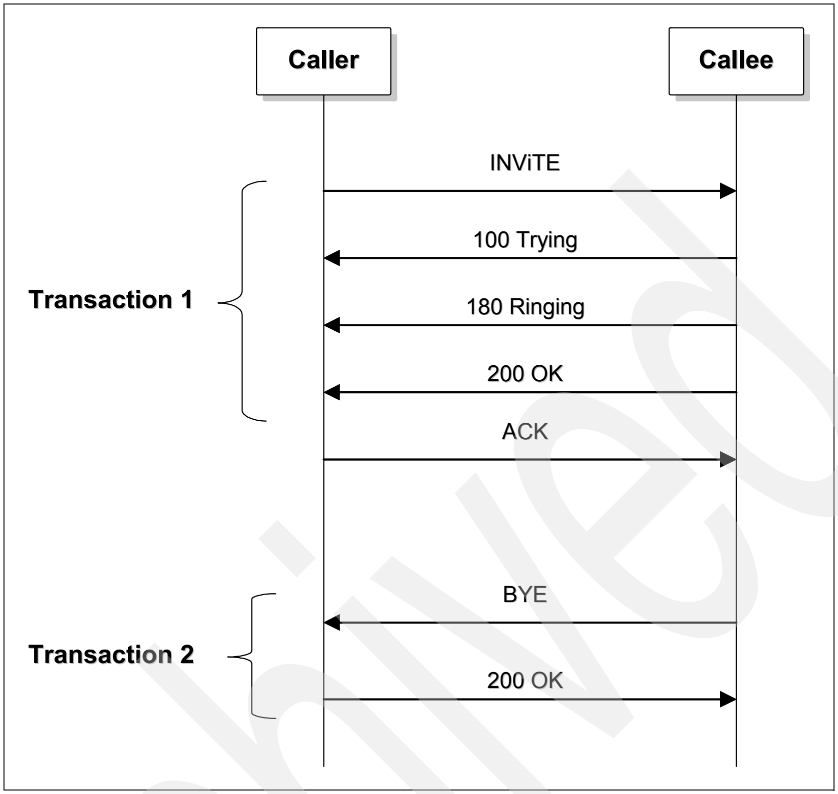


Figure 1-5 SIP transactions

1.3.4 SIP dialogs

SIP dialogs represent peer-to-peer relationships between two SIP endpoints. It provides the context for sequencing and routing of messages between SIP user agents. Dialogs are identified by the following: Call-ID header, and the From Tag and To Tag parameters. The value of these three fields are the same for messages that belong to the same dialog. The header field CSeq is used to sequence messages within a dialog. The value is increased monotonically from request to request, thereby identifying the transactions within a dialog. In effect, a dialog is a sequence of transactions. This is illustrated in Figure 1-6 on page 17.

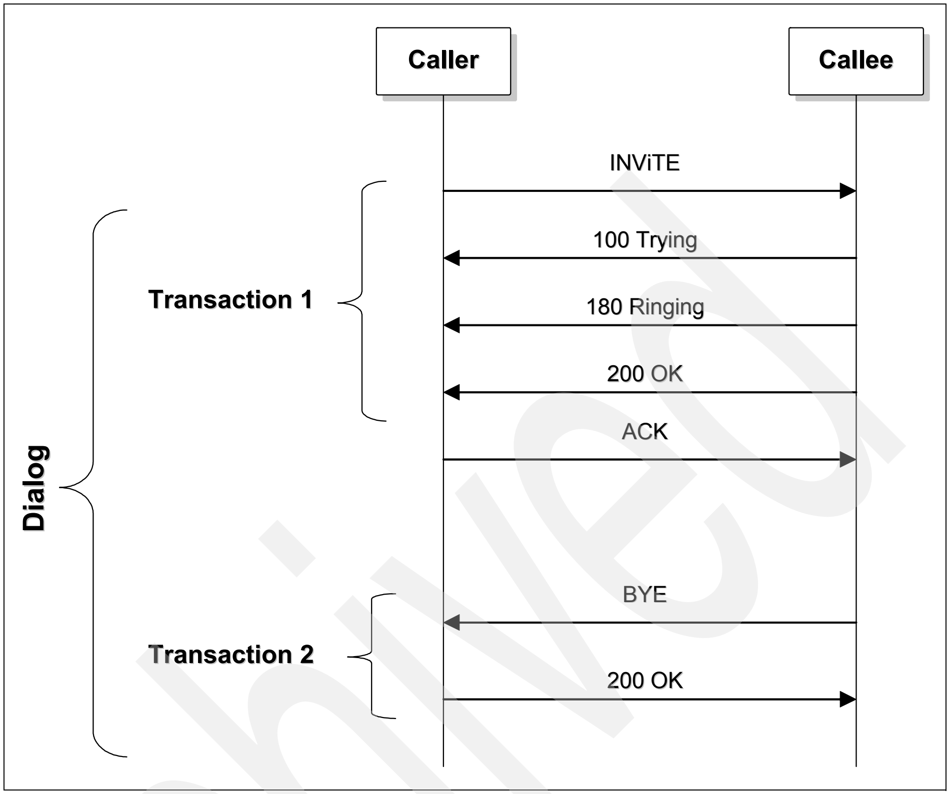


Figure 1-6 SIP dialog

1.3.5 A sample SIP call flow

Figure 1-7 on page 19 illustrates the message flow in a simple SIP call between two user agents. It shows a user Alice establishing a call with another user Bob. The steps in the following description correspond to the numbering in Figure 1-7 on page 19.

- Step 1 - 2

  This is the first request that a UA sends. It is a REGISTER message. It provides the server with an address at which Alice can be reached for SIP sessions. In this message Alice is registering herself, though it is possible that one SIP user can register on behalf of another user. As can be seen in the REGISTER message (and all other messages from Alice) each message goes through the Proxy Server.

- Step 3 - 4

  The response to the REGISTER message is positive, which is indicated by the 200 (OK) message.

- Step 5 - 6

  Alice sends a SIP INVITE to Bob. Let us assume that Alice doesn't know where Bob has registered, and hence the To address field in the SIP INVITE header is left blank.

- Step 7 - 8

  The Redirect Server responds with a status code 302 (moved temporarily). This response has field Contact in the header which is filled with an address where Alice should try as an alternative. Though not shown in the diagram, it is assumed that Bob has pre-registered before the call flow.

- Step 9 - 10

  Alice acknowledges (ACK) the 302 message.

- Step 11

  Alice now sends a new SIP INVITE message for Bob.

- Step 12

  The role of the proxy server for this message becomes crucial. The proxy server may map the Request-URI message to a different address, if it knows that the recipient is at a different address. In our case, Alice knows Bob address, and hence this won't be necessary. In this case, the Proxy server just inspects the domain part of the received Request-URI and determines the next hop in the path from the caller (Alice) to the callee (Bob). After the initial INVITE request and response, it is possible that subsequent messages are sent end to end (without traversing a Proxy Server). However, a proxy server can also ensure that it remains in the signaling path for all subsequent requests as well.

- Step 13

  The Proxy Server sends a Trying message. Any 1XX response is a provisional response and it indicates that a session has not yet been established, but a dialog is on. This is an early state of a dialog. There could also be an optional SIP 180 Ringing Message (Not shown in this call flow).

- Step 14 - 15

  Bob answers the call with a 200 OK response, indicating that he is ready to take a call.

- Step 16 - 17

  Alice sends an ACK to confirm that she has received Bob's OK response. This ends the three way handshake between the two parties. Once Alice sends the ACK, both parties are ready to exchange media.

- Step 18 - 19

  Any party can initiate a BYE request to terminate a session. In this case, Alice sends a BYE message to terminate the session.

- Step 20 - 21

  Bob responds to the completion of the call with an OK response to the BYE.

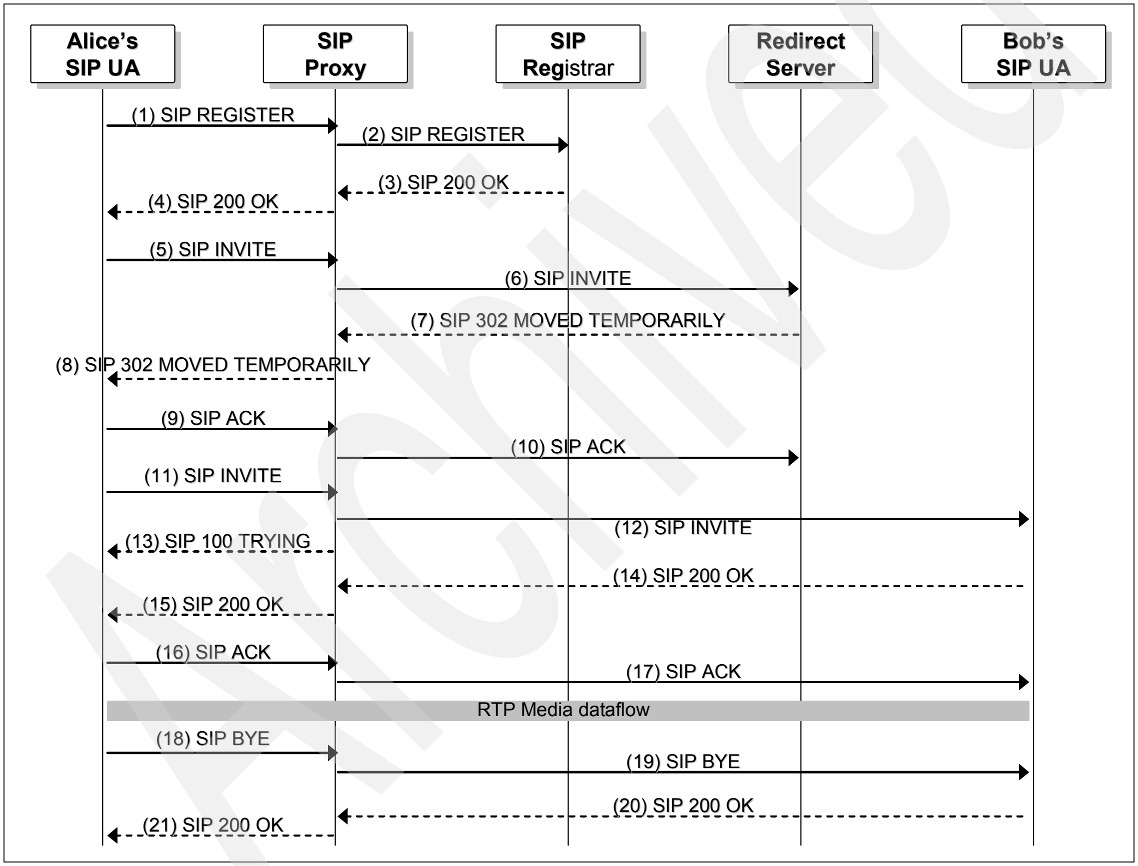


Figure 1-7 A simple SIP call flow

1.4 SIP Java development

The Java Community Process (JCP) through the Java APIs for Integrated Networks (JAIN™) initiative, define Application Programming Interfaces (API) for using Java technologies to provide next generation telecommunications services. Three of the APIs developed under JCP and JAIN initiative support SIP programming for call control, messaging, presence and location based services running on devices ranging from mobile handsets to application servers.

API	Java platform	Target
JAIN SIP	J2SE	Client
SIP Servlet	J2EE	Server Web Tier
SIP for J2ME	J2ME	Device

Figure 1-8 JCP SIP programming APIs

Note: SIP for J2ME™ API defines SIP interface for small devices that support J2ME platform and Mobile Information Device Profile (MIDP). The focus of this redbook is the development of server side SIP and IP Multimedia Subsystem applications, as a result, SIP for J2ME API is not discussed.

1.4.1 JAIN SIP API

The JAIN SIP architecture supports an asynchronous events and messaging model between Providers and Listeners. Messages are correlated into transactions and dialogs. The JAIN SIP API defines standardized interfaces to the stateless and stateful transaction, as well as the stateful dialog models defined by the SIP protocol. It enables application interoperability across SIP stack implementations through Java interfaces for:

- SIP stack
- Messaging
- Events

Note: See Figure 1-9 on page 22 for overview of the JAIN SIP architecture.

The interfaces are realized through Java interfaces SipStack, SipProvider and SipListener. They encapsulated SIP protocol functionality which can be utilized in the development of SIP user agents, SIP proxy servers, SIP registrars or embedded into SIP service containers.

- SipStack

  This interface defines the methods and view for representing and managing proprietary SIP stack. Included in the set of methods are the creation and deletion methods for SipProvider and ListeningPoint used by applications that implement SipListener interface.

  The JAIN SIP API specification defines a single SipStack object for a given IP address. However there is a one-to-may relationship between SipStack and a SipProvider as well as a ListeningPoint.

- SipProvider

  The SipProvider interface defines the messaging and transaction view of the SIP stack. The set of methods provide the following functionality which applications that implement SipListener rely on:

  – Registration - a SipListener must register with the SipProvider in order to be notified of Events representing either Request, Response or Timeout messages

  – Deregisteration - by deregistering, a SipListener will stop receiving Events from the SipProvider

  – Send stateless Request or Response

  – Transaction creation

  – Listening Point manipulation

  – SipStack object accessor

- SipListener

  This interface is implemented by applications that use the JAIN SIP API. It presents the application with a view of the SIP stack and defines the channel for the application to receive and process Events from the SipProvider. Three types of Events are emitted to the application through the SipListener interface:

  – RequestEvent - are request messages such as INVITE that are received from the network and passed up the SIP stack to the application

  – ResponseEvent - are response messages such as 2xx (200 OK) that are received from the network and passed up the SIP stack to the application

  – TimeoutEvent - are emitted by SipProvider and represent need to retransmit the transaction that timed out

- ListeningPoint

  This interface represents the port that SipProvider uses for sending and receiving messages

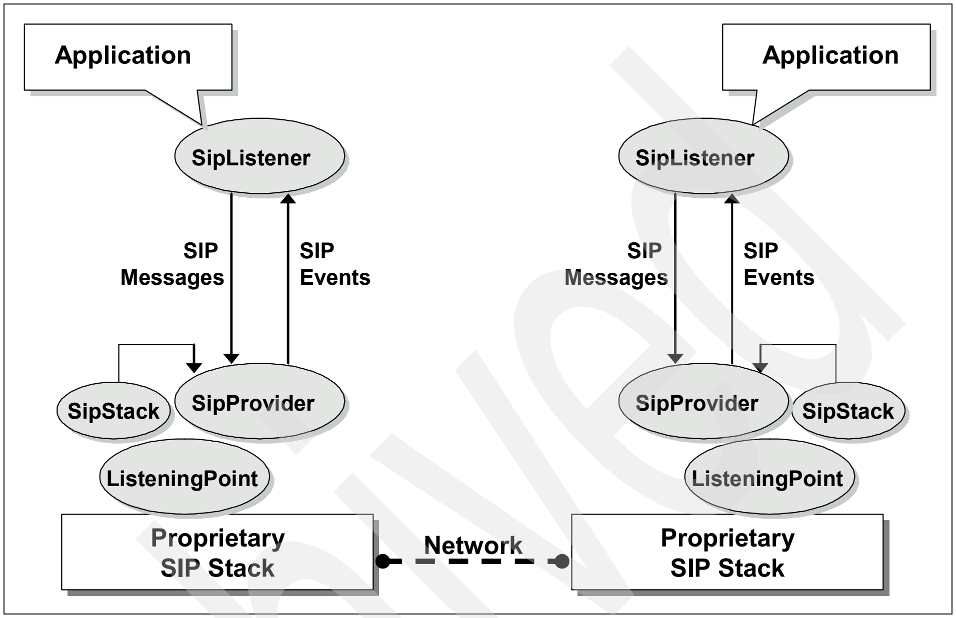


Figure 1-9 JAIN SIP architecture overview

> **Note:** More information about the JAIN SIP API specification can be obtained from the Java Community Process Web site at:
>
> http://jcp.org/en/jsr/detail?id=32

JAIN SIP supports the SIP protocol functionality as described in RFC 3261 “SIP: Session Initiation Protocol”

JAIN SIP also supports the following SIP extensions:

- RFC 2976

  The SIP INFO Method - this extension adds the INFO method to the SIP protocol

- RFC 3262

  Reliability of Provisional Responses in the Session Initiation Protocol - this extension uses the option tag 100rel and defines the Provisional Response ACKnowledgement (PRACK) method

- RFC 3265

  SIP Specific Event Notification - this extension provides the

  framework for requesting notification when certain events occur

- RFC 3311

  Session Initiation Protocol (SIP) UPDATE Method - this extension allow for the updating of session parameters prior to the final response

- RFC 3326

  The Reason header field for SIP - the addition of this field enable the ability to know why a SIP request was issued

- RFC 3428

  SIP extension for instant messaging - this extension introduces the MESSAGE method to allow for the transfer of Instant Messages

- RFC 3515

  The SIP Refer method - this extension requests that the recipient refer to a resource provided in the request

1.4.2 SIP Servlet API

SIP Servlet specification was developed through the Java Community Process (JSR 116). It presents an abstract view of SIP that is based on the Java servlet API. SIP Servlets are Java based component applications that typically run in servlet containers on network servers. Where JAIN SIP API provides access to the full SIP protocol, the SIP servlet and container hide SIP protocol complexities by providing an environment where services are prevented from violating the protocol or performing restricted operations.

The SIP container performs many of the same functions in order to simplify the creation of SIP applications. Servlet container manages the life cycle of a servlet and support interaction between servlets and SIP clients (user agents) by exchanging SIP request and response messages. For instance, a servlet container can perform message queuing, dispatching and state management.

Like HTTP servlets, SIP Servlets extend base javax.servlet.GenericServlet class. The SipServletRequest and SipServletResponse classes are similar to the HttpServletRequest and HttpServletResponse classes. All messages come in through the service method which calls `doRequest` or `doResponse` for incoming requests and responses, respectively. Depending on the request method or status code the call is dispatched to one of the following methods:

Table 1-4 SIP servlet request /response methods

Method	Request/Response message
Requests	
doInvite	SIP INVITE requests
doAck	SIP ACK requests
doOptions	SIP OPTIONS requests
doBye	SIP BYE requests
doCancel	SIP CANCEL requests
doRegister	SIP REGISTER requests
doSubscribe	SIP SUBSCRIBE requests
doNotify	SIP NOTIFY requests
doMessage	SIP MESSAGE requests
doInfo	SIP INFO requests
doPrack	SIP PRACK requests
Responses	
doProvisionalResponse	SIP 1xx informational responses
doSuccessResponse	SIP 2xx responses
doRedirectResponse	SIP 3xx responses
doErrorResponse	SIP 4xx, 5xx, and 6xx responses

> **Note:** Despite being derived from a common base class, there are differences between SIP Servlets and HTTP Servlets. While HTTP is a synchronous request/response protocol, SIP is asynchronous, and there is not necessarily one response, or any responses to every SIP request dispatched to a SIP Servlet. Also, SIP programming model allow applications to act as a client or proxy to other servers or proxies.

The SIP Servlet API includes a set of objects and interfaces that provide high-level abstraction of many of the SIP concepts. Table 1-5 lists the key items in the API.

Table 1-5 SIP servlet objects and interfaces

Interface/Object	Description
`SipServlet`	The base servlet object, it receives incoming messages through the `service` method, which calls `doRequest` or `doResponse`.
`ServletConfig`	Used by the servlet container to pass configuration information to a servlet during initialization.
`ServletContext`	Used by a servlet to communicate with its container.
`SipServletMessage`	Defines common aspects of SIP requests and responses.
`SipServletRequest`	Provides high-level access to SIP request messages. Created and passed to the handling servlet when the container processes incoming requests.
`SipServletResponse`	Provides high-level access to a SIP response message. Instances of `SipServletResponse` are passed to servlets when the container receives incoming SIP responses.
`SipFactory`	Factory interface for a variety of servlet API abstractions.
`SipAddress`	Represents SIP From and To header.
`SipSession`	Represents SIP point-to-point relationships, and maintains dialog state for UAs.
`SipApplicationSession`	Represents application instances, acts as a store for application data and provides access to contained protocol sessions.
`Proxy`	Represents the operation of proxying a SIP request and provides control over how that proxying is carried out.

SIP servlet supports the baseline SIP protocol functionality as described in RFC 3261 “SIP: Session Initiation Protocol”.

SIP servlet also supports the following SIP extensions:

- RFC 3265

  Session Initiation Protocol (SIP)-Specific Event Notification - this extension provides the framework by which SIP nodes can be notified of events that occur at remote nodes.

- RFC 3428

  SIP extension for instant messaging - this extension introduces the MESSAGE method to allow for the transfer of Instant Messages.

1.5 Examples of SIP application

One application area that showcases the use of SIP is the area of subscription and notification for events such as Presence. This in turn has applications in Instant Messaging. The IETF working group - SIP for Instant Messaging (IM) and Presence Leveraging Extensions (SIMPLE) was set up to address the use of SIP for messaging and presence. SIP was extended to also support SIP MESSAGE method for use in instant messaging where each message is independent of any other message. In addition to applications related to Presence and IM, the convergence of SIP servlet and HTTP servlet containers is enabling applications that leverage SIP, such as “click-to-talk”, which integrates communication and conferencing into existing enterprise applications.

2

Introduction to IP Multimedia Subsystem

In this chapter we introduce the concept, architecture and standards that is called IP Multimedia Subsystem (IMS). We also provide an overview of IMS as the catalyst for convergence and enabler for service-driven development and delivery of new applications.

This chapter describes the following:

- IMS overview
- Elements of IMS architecture
- Services in IMS

2.1 IMS overview

IP Multimedia Subsystem (IMS) is a set of requirements and specifications defined by 3rd Generation Partnership Project (3GPP) and 3rd Generation Partnership Project 2 (3GPP2). 3GPP and 3GPP2 were formed through collaboration agreements that include a number of regional telecommunication standards bodies. IMS defines a unifying architecture for IP-based services over both packet- and circuit-switched networks. It enables the convergence of different wireless and fixed access technologies for the creation, delivery and consumption of multimedia services. IMS also supports services integration through standardized reference points (akin to interfaces and protocols) which not only makes service creation faster and easier but also leverages the services available through Internet technologies such as Web Services.

2.1.1 IMS vision and history

IMS is at the center of the 3G (third Generation) initiatives driven by 3GPP and 3GPP2 who are participants in International Mobile Telecommunications-2000 (IMT-2000) the global standard for third generation (3G) wireless communications, defined by a set of interdependent International Telecommunications Union (ITU) Recommendations.

The 3GPP was formed as a collaboration of many organizations (includes Association of Radio Industries and Businesses, China Communications. Standards Association, European Telecommunications Standards Institute, Alliance for Telecommunications Industry Solutions, Telecommunications Technology Association of South Korea and Telecommunication Technology Committee of Japan) in 1998 to develop the technical specifications for a 3G network evolving from a GSM network. A similar charter, 3GPP2 (includes Association of Radio Industries and Businesses, Telecommunication Technology Committee of Japan, China Communications. Standards Association, Telecommunications Technology Association of South Korea and Telecommunications Industry Association), was formed to evolve the North American and Asian networks from traditional CDMA2000 networks into a 3G platform.

Both 3GPP and 3GPP2 conceptualized their respective IMS (IP Multimedia Subsystem) architectures to support packet switched communication, in order to merge the Internet and the cellular worlds. The IMS core network has a common IP based transport and signalling, which can be accessed by different networks. IMS common signalling is based on SIP. We presented an overview of SIP in Chapter 1, “Introduction to Session Initiation Protocol (SIP)” on page 3 as an end-to-end based session management control protocol. SIP allow applications to remain agnostic of the access network, which matches the network access requirements for IMS.

Initial concepts for IMS emerged with the Universal Mobile Telecommunications System (UMTS) third-generation (3G) specifications in 1998. The first specification of IMS was published in March of 2003 by 3GPP in UMTS Release 5. UMTS Release 5 provided general description of IMS, SIP and end-to-end Quality of Service (QoS) as part of an “All IP” network. IMS continues to evolve with each UMTS release since 2003. New functions and changes to IMS are introduced through 3GPP approved change requests (CRs) with each release. 3GPP release 6 and 7 added interworking with wireless local area networks (WLAN) and support for fixed networks, by working together with TISPAN (Telecoms & Internet converged Services & Protocols for Advanced Networks).

> **Note:** Telecoms & Internet converged Services & Protocols for Advanced Networks (TISPAN) is a standardization body that is working through the European Telecommunications Standards Institute (ETSI) to define the Next Generation Network architecture. Some aspects of the TISPAN Release 1 architecture is based on IMS.

The vision of IMS as the common platform for development and delivery of diverse multimedia services for a true mobile Internet is based on the set of requirements set forth in the 3GPP IMS requirements captured in 3GPP TS 22.228.

Available at:

http://www.3gpp.org/ftp/Specs/html-info/22228.htm

The following are the highlights of the key IMS requirements:

- IP multimedia sessions

  IP multimedia session requirements are focused on the main service delivered by IMS, it includes support for a variety of media (such as voice, video and data) and one or more applications that provide the service experience within sessions. The IP multimedia session requirements call for media interoperability and per user application flow control.

- Quality of Service (QoS)

  IMS QoS characteristics are negotiated between end points in IMS sessions. The parameters include the type of media, the codecs and encoding formats, bandwidth, delay, delay variation and packet loss. The IMS QoS requirements include negotiations at session establishment and during the session as well as for individual media components. IMS requirements also allow for operators to set policy and control QoS for all or individual users.

- Interworking

  Interworking requirements support the principle of access independence, where subscribers can access IMS services regardless of how they obtained IP connection. Interworking requirements also cover interwork between circuit-switched and cellular networks.

- Roaming

  The IMS roaming requirements are inherited from the second generation cellular networks where users are able to roam in different networks and access services provisioned by the user's home environment or the servicing network. Roaming requirements include support for automated negotiation between operators for QoS and service capabilities.

- Service creation

  This is one of the key IMS requirements. The service creation requirements stresses standardization of service capabilities instead of services. By not requiring standardized services, IMS enable flexibility in service creation while eliminating the significant delays from interoperability and testing constraints of standardized services.

2.2 Elements of IMS architecture

The IMS architecture follows a functional approach rather than a physical one. The functional architecture consist of separate functional components with standardized interfaces between the components. In other words, it defines the functions that need to exist and not the physical boxes or nodes where each function should reside. The decision as to what functions would reside in which node and how to combine several functions into one node or split functions across several nodes is implementation dependent.

Figure 2-1 on page 31 is an illustration of the IMS core network reference architecture, showing the functional components and the standardized interfaces between the components that are referred to as "reference points".

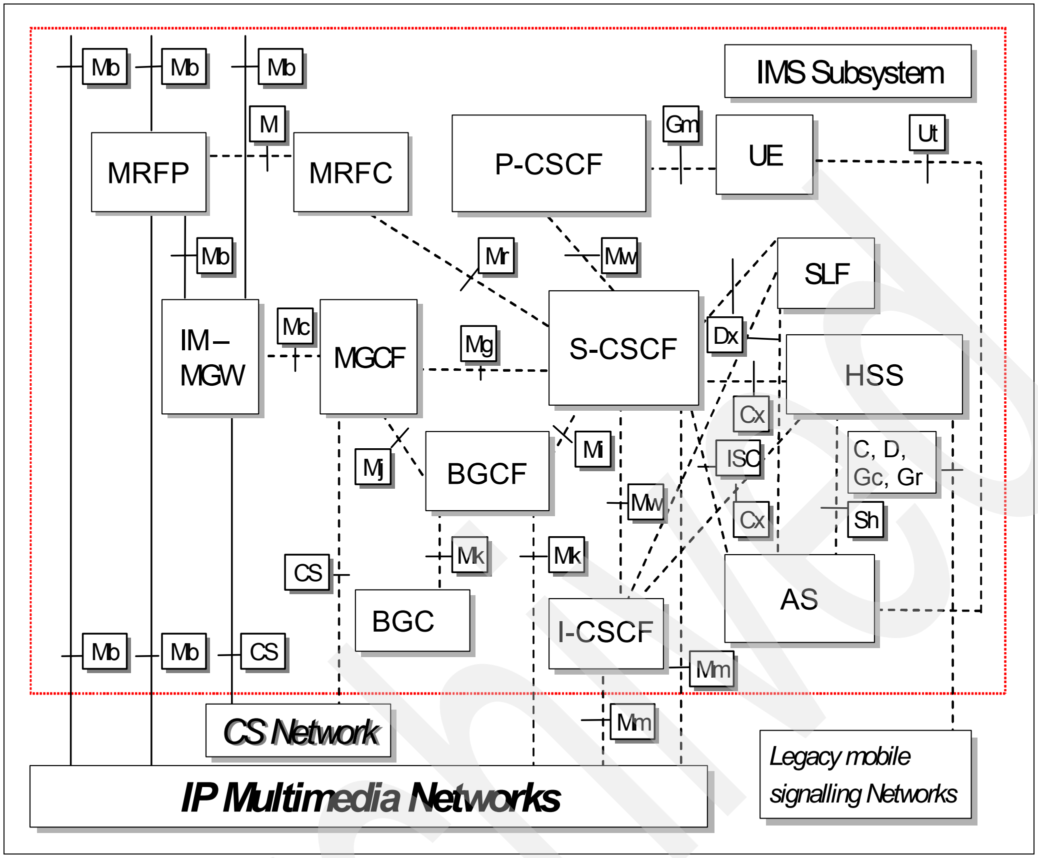


Figure 2-1 IP Multimedia Core Network Subsystem reference architecture (Source 3GPP)

2.2.1 Functional components

The IP Multimedia Core Network Subsystem include the different functional components for a network infrastructure for delivering multimedia services. The components include databases for maintaining subscriber information, call and session control components, media and application servers, media/signalling gateways and user equipments for accessing the network.

- IMS Terminals

  They are typically referred to as User Equipment (UE). Examples of UEs are mobile phones, PDAs (Personal Data Assistant) and computers. Though the figure shows the UEs accessing the core network through a radio network, in effect (since IMS supports an access agnostic architecture), they can access the IMS core through other access topologies, such as WLAN.

- User Databases

  The Home Subscriber Server (HSS) contain user data related to subscription of services. The user data include user profile, location information and security information. In case a network contains more than one HSS, another database known as Subscriber Location Functions (SLF) maps the users' address to the HSS where the data is stored.

- SIP Servers

  These are also known as CSCF (Call/Session Control Functions). They perform session control functions for IMS sessions. CSCFs can be categorized into three groups, based on the functionality:

  – Serving CSCF (S-CSCF)

    An S-CSCF is the hub of all signaling functions in an IMS network. In addition to session management, an S-CSCF also performs the role of a SIP Registrar within an IMS network. There is a Diameter interface between S-CSCF and HSS/SLF for downloading authentication data and user profile. All incoming/outgoing messages to/from a UE traverses the allocated S-CSCF and it inspects these messages for necessary actions that need to be taken (for example to authorize a user for a particular action, based on the user profile). Based on the message, an S-CSCF performs routing functions. Routing needn't just involve routing messages to another SIP Server, but could involve Application Servers as well. The user profile (downloaded by the S-CSCF from the HSS) actually instructs an S-CSCF whether the SIP signaling message should be routed to one or more Application Servers before it is routed to the final destination.

  – Proxy CSCF (P-CSCF)

    P-CSCF performs the role of a SIP Proxy Server for inbound and outbound messages from an IMS Terminal UE. Once a UE registers itself in the network, it is assigned a P-CSCF for the duration of the registration. The tasks performed by a P-CSCF are similar to what we discussed for a SIP Proxy Server in 1.2, “SIP architectural components” on page 7, with regards to authentication, security and validation of SIP messages. It also generates usage/charging data for the UE.

  – Interrogating CSCF (I-CSCF)

    Strictly speaking an I-CSCF is also a SIP Proxy Server. However, its location and function is specific in nature. It is located at the edge of an administrative domain of a network. When a P-CSCF wants to find the next hop for a SIP message, it obtains the address of the I-CSCF of the destination network. The I-CSCF uses its Diameter interface with the HSS/SLF to find the S-CSCF assigned to the UE. It subsequently forwards the incoming SIP message to the appropriate S-CSCF.

- Application Servers

  Application servers essentially host and execute services for users and perform the function of SIP Application Servers. In other words, depending on the actual service, application server can operate in of the following modes:

  – SIP Proxy mode
  – SIP User Agent mode
  – SIP Redirect Server mode
  – SIP B2BUA (Back to Back User Agent - concatenation of two user agents)

  Application servers also interface with the HSS to upload and download user data.

- Gateways

  Several types of gateways are supported in the IMS architecture, for example the architecture includes gateways for converting signal from packet switched IMS network to a circuit switched PSTN or vice versa

  – Signaling gateways performs lower layer protocol conversion from one network to another.

  – Gateways to convert the media data - Media Gateway (MG) and Media Gateway Controller Function (MGCF). The MG interfaces the media planes of two networks. Hence, it converts the media over RTP (in IMS network) to the PCM (Pulse Code Modulation) based transport in the PSTN side.

- BGCF (Breakout Gateway Controller Functions)

  The BGCF is also a SIP server that performs routing functions when the call is addressed to a circuit switched network such as PSTN. It locates the appropriate gateway at the circuit switched destination network for routing the outgoing call.

- Media Resource Function (MRF)

  An MRF performs several media functions for the home SIP network, such as mix media streams (in a conference bridge), transcoding functions, playing announcements. An MRF can be further broken into MRF Controller (MRFC) and MRF Processor (MRFP). The MRFC acts essentially as a SIP UA interfacing with the S-CSCF) to manage resources of the MRFP, while the MRFP performs all the media functions stated above.

2.2.2 Reference points

Performing functions in IMS network is realized through procedures which define the flows between functional components. The interfaces exposed by the functional components and the control between the components is referred to as "reference points".

Note: For the technical specifications of reference points in the 3GPP network architecture, refer to the specification “3GPP TS 23.002” at:

`http://www.3gpp.org/ftp/Specs/html-info/23002.htm`

The following is the description of the reference points for the IP Multimedia Core Network Subsystem.

- Cx - Supports information transfer between CSCF and HSS
- Dx - The CSCF and SLF interface is used to retrieve the address of the HSS which holds the subscription date for a given user. Not required in a single HSS environment
- Gm - Supports communication between the UE and a P-CSCF
- ISC - The interface between the CSCF and application servers for access to IMS services
- Ma - The interface between an application server and an I-CSCF
- Mb - Used to access IPv6 network services for user data transport
- Mg - Allows the MGCF to forward incoming session signalling (from the PSTN) to the CSCF for the purpose of interworking with PSTN networks. Uses SIP for signalling
- Mi - Allows the Serving CSCF to forward the session signalling to the BGCF for the purpose of interworking with PSTN networks. Uses SIP for signalling
- Mj - Allows the BGCF to forward session signalling to the MGCF for the purpose of interworking with PSTN networks. Uses SIP for signalling
- Mk - Allows the BGCF to forward session signalling to another BGCF. Uses SIP for signalling
- Mm - The interface between a CSCF/BGCF/IMS ALG and an IP multimedia network
- Mr - Supports information transfer between CSCF and MRFC. Uses SIP for signalling
- Mw - Allows the communication and forwarding of SIP signalling messaging between CSCFs
- Mx - The interface between a CSCF/BGCF and IBCF
- Sh - Used for communication from the SIP or OSA application server to the HSS
- Si - Used for communication from the CAMEL application server to the HSS
- Ut - The Ut interface resides between the UE and the SIP Application Server

2.2.3 Protocols

IMS works with a number of protocols. In designing IMS protocols, 3GPP leveraged the work of other Standards Development Organizations (SDOs) such as the IETF and ITU-T by reusing existing protocols. The following are some of the protocols:

- SIP - Session Initiation Protocol

  The Session Initiation Protocol (SIP) (RFC3261) defined by IETF is the chosen as the session control protocol for the IMS.

- DIAMETER

  The Diameter protocol provides an Authentication, Authorization, and Accounting (AAA) framework. The Diameter Base Protocol defined in IETF RFC 3588 stipulates the minimum requirements for an AAA protocol. IMS uses Diameter protocol to provide the necessary authentication, authorization, and, for billable communications, accounting services.

- COPS - Common Open Policy Service

  The Common Open Policy Service (COPS) protocol (RFC 2748) is a query and response protocol for exchanging policy information. This IETF protocol is used for communication of QoS within the IMS architecture.

- H.248/MEGACO - Media Gateway Control

  H.248 is an International Telecommunications Union Telecom Standardization Sector (ITU-T) standardized Gateway Control Protocol (GCP). The Internet Engineering Task Force (IETF), endorses this protocol and refers to it as Megaco (Media Gateway Controller). H.248 is used by IMS Media Gateway (MG) for media conversion provide end-to-end communication.

- RTP/RTCP - Real-Time Protocol / Real-Time Control Protocol

  RTP is used in IMS as the media protocol for end-to-end delivery of services. RTCP is used for feedback on the transmission and reception quality of data carried by RTP. IMS relies on these protocols for transfer of real time media such video and audio.

- SCTP - Stream Control Transmission Protocol

  SCTP is designed to transport PSTN signaling messages over IP networks. As a reliable transport protocol, SCTP has other applications such as delivery mechanism for multimedia (SIP) and for wireless.

Note: The list of IMS protocols above are just the key protocols supported by IMS. A complete protocol stack needed for IMS includes many more protocols.

2.2.4 Functional planes

The 3GPP architecture consist of logical planes or layers which correspond to discrete functions. This is one of the most powerful concepts of the IMS functional architecture. Each plane consist of IMS functional components that together provide the functions supported by the plane. The logical functions in IMS are divided into the following three planes (see Figure 2-2 on page 37 for illustration):

- Transport plane

  The transport plane provides support for the backbone IMS network and for the different means through which users can gain access to the IMS network. Included in this plane are IMS components such as routers, media gateways and switches as well as IMS user equipment devices. These components translate protocols between the IMS core network and the connecting network.

  The transport plane also shields the upper layers of the IMS architecture from the network access technologies by providing common access interface to the components in this planes.

- Control plane

  The primary function of the Control plane is to provide switching and session control in IMS networks. The key components in this plane are the SIP servers and proxies collectively called Call/Session Control Function (CSCF). CSCF handles SIP registrations and routing of the SIP signaling messages to appropriate application servers amongst the other control and signalling functions that it performs. CSCF also provides policy control and QoS management.

  The other components in this plane include:

  – Home Subscriber Server (HSS)

    The repository for users service profile. The user information is also used to provide authentication, authorization and accounting (AAA) functions.

  – Media Gateway Control Function (MGCF)

    MGCF interworks SIP signaling with the signaling used by the media gateways. and manages the connections between the PSTN and the IP streams. It converts SIP messages into either Megaco or ISUP messages.

  – Media Server Function Control (MSFC)

    The MSFC provides a similar function as the MGCF for media servers

- Service plane

  Residing in the service plane are application servers that perform telephony and non-telephony functions. Application servers interface with Call Session

Control Function in the control plane using SIP, and operate in SIP proxy, SIP User Agent Server (UAS) or SIP B2BUA (back-to-back user agent) mode. An AS can be located in the home network or in an external third-party network.

Types of application servers include:

– SIP AS - application servers that interface using SIP

– OSA-SCS (Open Service Access - Service Capability Server) - interfaces with Open Services Architecture (OSA) application servers using Parlay

– IM-SSF (IP Multimedia Service Switching Function) - CAMEL application servers, interfaces using CAMEL Application Part (CAP)

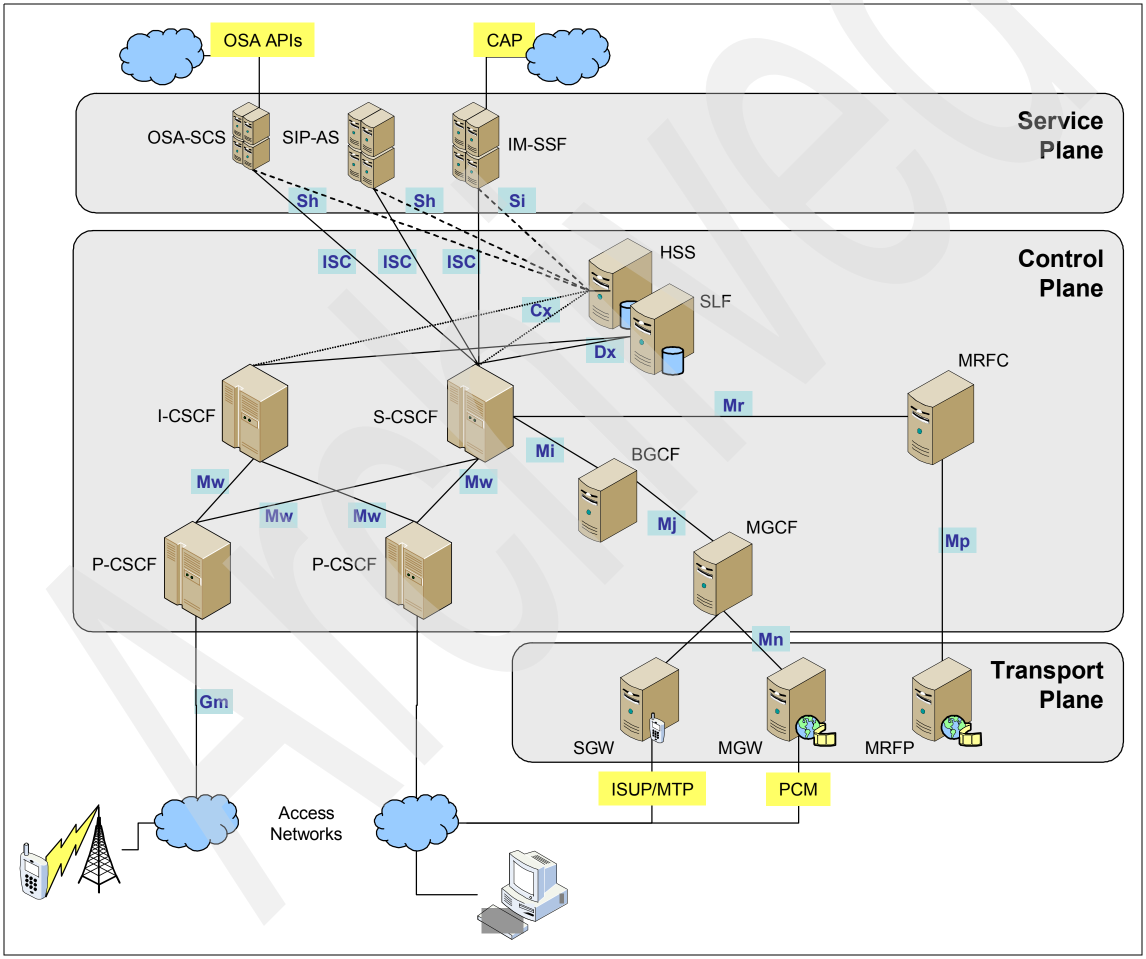


Figure 2-2 IMS functional planes

2.3 Services in IMS

One of the promises and objectives of IMS is the ability to develop and deploy services as quickly as possible. The IMS architecture is designed to enable this capability by providing an environment that is in contrast to the traditional vertically integrated silo network environment that supported individual services. The single converged network environment created by IMS eliminate multiplicity of services by enabling sharing of services across the different functionality planes thereby reducing cost and creating better user experience.

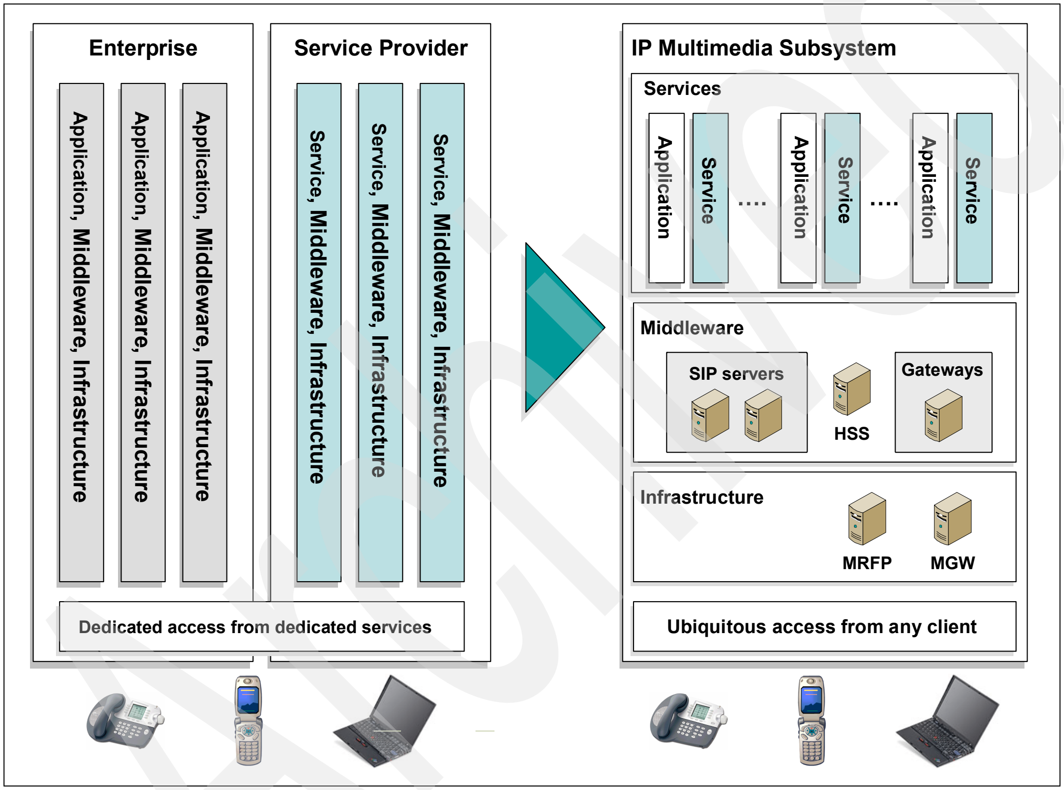


Figure 2-3 The IMS integrated services environment

2.3.1 Service architecture

At the center of IMS services architecture are a combination of the servers in the services plane (discussed in 2.2.4, “Functional planes” on page 36). These include servers that interface with the Call Session Control Function, other functions in the control plane and gateways to services, feature specific application servers that function as service enablers, and finally third party application servers that enable creation and composition of converged services.

One of the key IMS requirements is standardization of service capabilities instead of services. While there are no standardized services, the servers in the service plane provide standardized services. Examples are the service enablers, such as group list management service, presence service, location service and charging service.

Figure 2-4 on page 40 presents an overview of IMS services architecture showing different types of servers and services. The servers support the full service life cycle from creation to delivery and execution and also part of the IMS service creation, deployment and delivery environments.

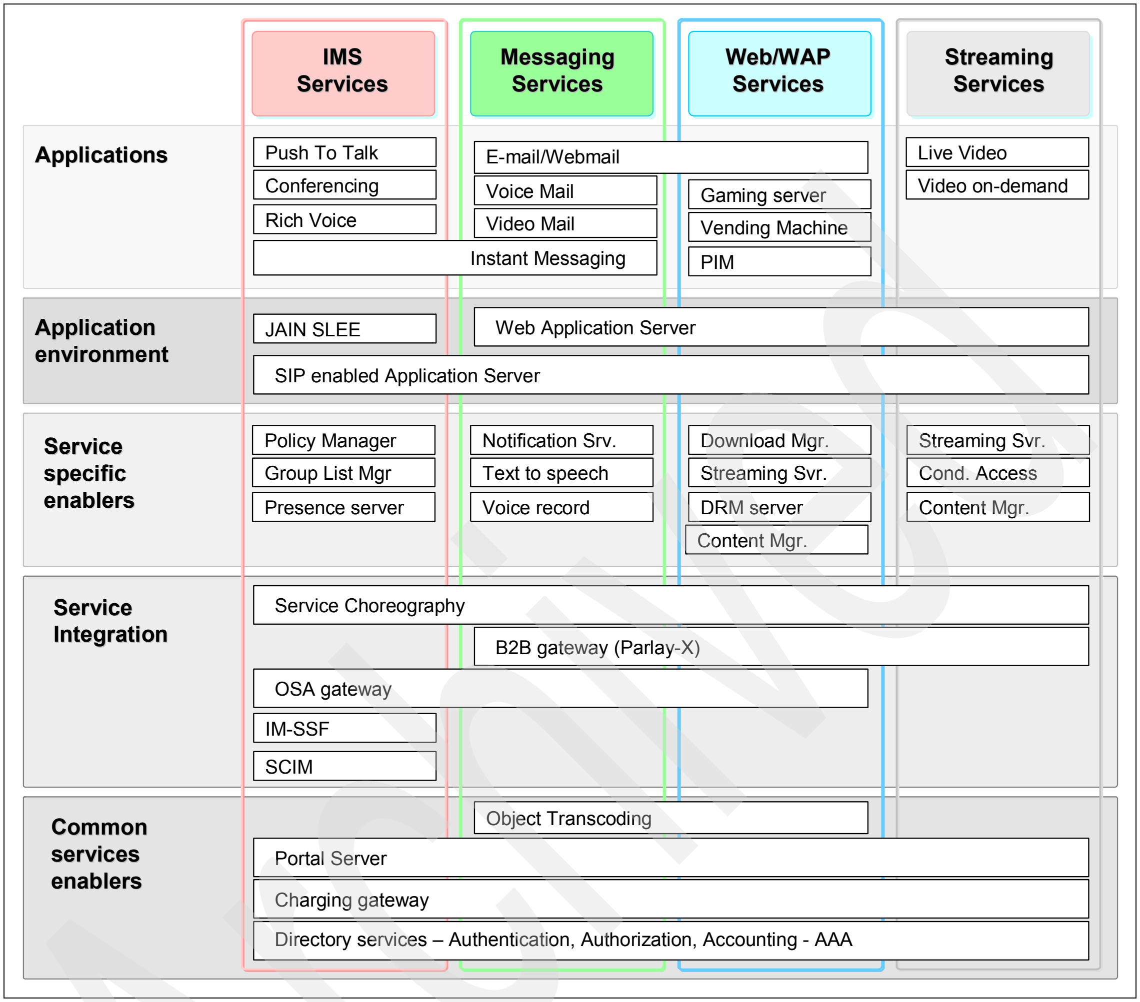


Figure 2-4 Overview of IMS services architecture

Part 2

Application development technologies

Part 2 provides an overview of the IBM service creation and execution environment. It highlights the features of the different tools that are used for creating SIP, converged SIP and HTTP, IMS foundation and IMS composite applications.

3

Introduction to IBM SIP and IMS service creation

This chapter introduces the IBM service creation environment and the different tools for developing SIP and IMS applications for the IBM WebSphere Platform for Telecom.

This chapter contains the following:

- Overview
- IBM Unified Service Creation Environment
- Types of SIP and IMS applications
- The SIP and IMS service creation environment
- The service execution environment

3.1 Overview

The real payoff of the IMS architecture and IP convergence lies in the ability to rapidly develop and deploy new services. The critical enabler is a robust service creation and delivery environment. While technology is central to service creation and delivery, it is only part of the equation. Developing high-quality, robust services and getting them to market on time require full cycle service development.

Full cycle service development brings together the business team that is responsible for setting the business goals and establishing the business priorities, the software development team that is responsible for developing new services and the operations team that is responsible for deploying and running the services.

Full cycle service development includes the following steps:

- Model the service

  Captures the service activities and flows, and also simulates alternative scenarios.

- Analyze requirements

  This next step defines the service requirements. By modeling the user interactions and precisely understanding the different flows, the business and technology requirements are captured.

- Design and construct

  The development team translates the requirements into technology solutions, using appropriate development paradigm to create high-quality services.

- Test

  Quality assurance activities ensure that software artifacts that are produced functions as designed with acceptable performance.

- Deploy

  Coordinated and managed deployment of services to the service execution environment.

- Monitor

  The performance-based feedback cycle that enable comparison of the projected value to actual results, and for making the necessary adjustments to maximize business returns.

These steps enable the three different constituencies, business, development and operations to work together in discovering business and technology assets, defining and prioritizing the requirements, developing at the speed of business and deploying in a closed loop. Monitoring the feedback lead to further discovery and resumption of the cycle. The cycle is illustrated in Figure 3-1.

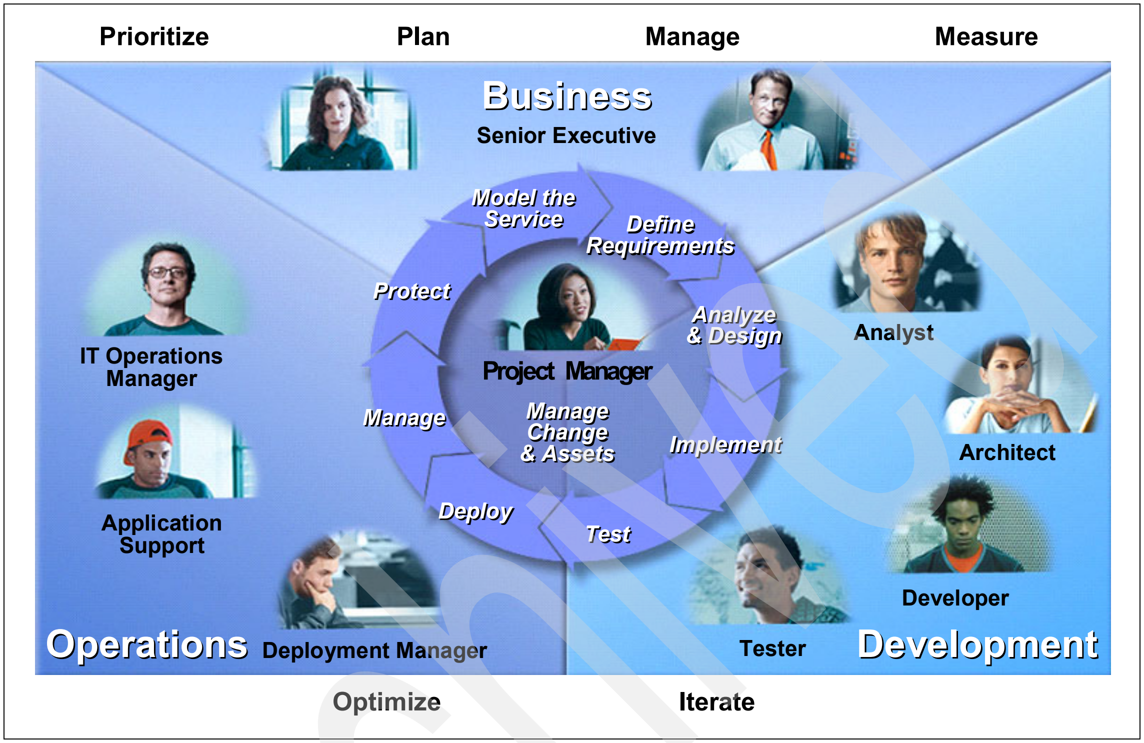


Figure 3-1 Full service development overview

3.2 IBM Unified Service Creation Environment

Recognizing the strategic importance of a robust service creation and delivery environment, IBM is developing the “IBM Rational® Unified Service Creation Environment” (USCE) to support full cycle service development.

USCE is a combination of a comprehensive set of tools, processes that capture proven best practices, and professional services designed to enable business driven services development. USCE provides tools that support the different teams and roles in the creation of services.

The technological underpinnings of USCE is the Eclipse Integrated Development Environment (IDE), the open source workbench with broad industry support. It provides a powerful and yet flexible tool integration infrastructure on which IBM has created its next-generation software tools platform.

Eclipse platform performs three primary functions in USCE:

- It provides the UI framework for a visually consistent rich client experience as you move between activities within USCE.
- It supports the sharing of information across different activities through use of a common set of models expressed in the Eclipse Modeling Framework (EMF) technology.
- Its integration infrastructure enabled the creation of teaming capabilities available throughout USCE.

USCE leverages the flexibility of the Eclipse framework to provide user interfaces that enable users to work in an environment that is tailored to their specific roles and the development tasks being performed. The use of a common set of models in the infrastructure makes it easy for the different roles to share artifacts across different activities.

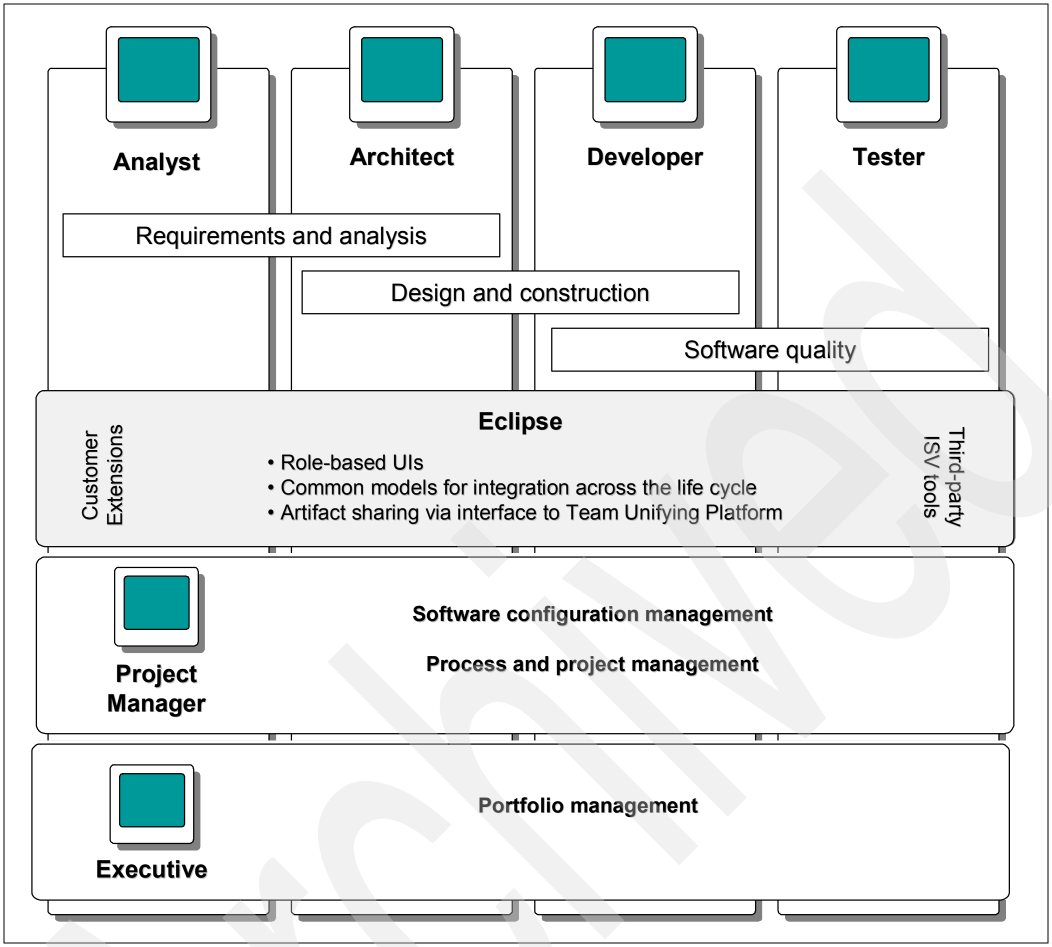


Figure 3-2 Roles in USCE

USCE include tools that provide support for all roles in the software development life cycle. The tools map to the following major areas of the software development life cycle:

- Requirements and analysis

  Integrated tools for requirements management, use case development, business modeling, and data modeling.

- Design and construction

  Tools for architecture and design modeling, model-driven development, component testing, and runtime analysis activities.

- Software quality

  Tools that address the three dimensions of software quality: functionality, reliability, and performance.

- Software configuration management

  Solutions for simplifying and managing change including version control, software asset management, and defect and change tracking.

- Process and portfolio management

  Integrated solutions that help teams manage change and requirements, implement a proven development process, and assess and report progress.

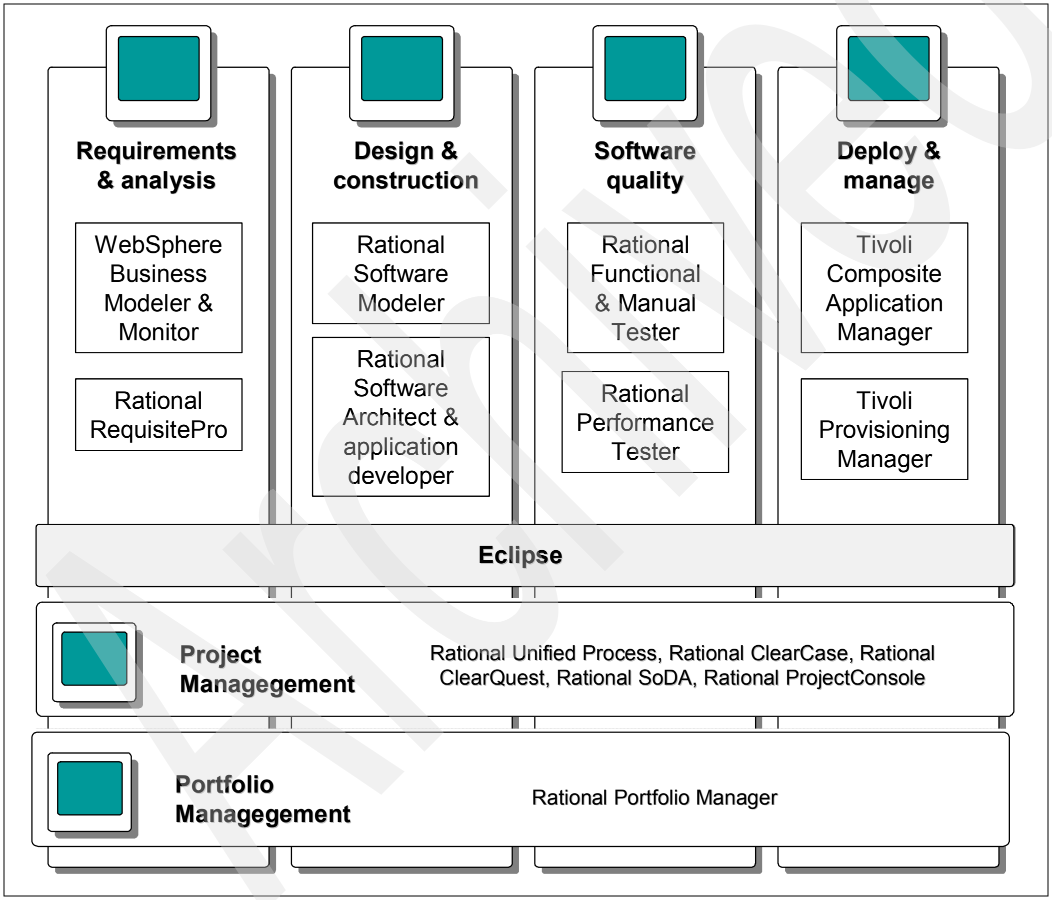


Figure 3-3 Overview of USCE portfolio

3.3 Types of SIP and IMS applications

Using toolkits in the USCE, you can create a number of distinct types of SIP and IMS applications. The types of applications include the following:

- SIP servlet applications that run in J2EE containers.
- Converged SIP applications that include SIP and HTTP servlets running in shared sessions in J2EE containers.
- IMS foundation applications where SIP servlet applications access IMS components using IMS Service Control (ISC), Diameter and XML Configuration Access Protocol (XCAP) interfaces.
- IMS composite applications where Business Process Execution Language (BPEL) processes implement the service logic and orchestrate foundation services

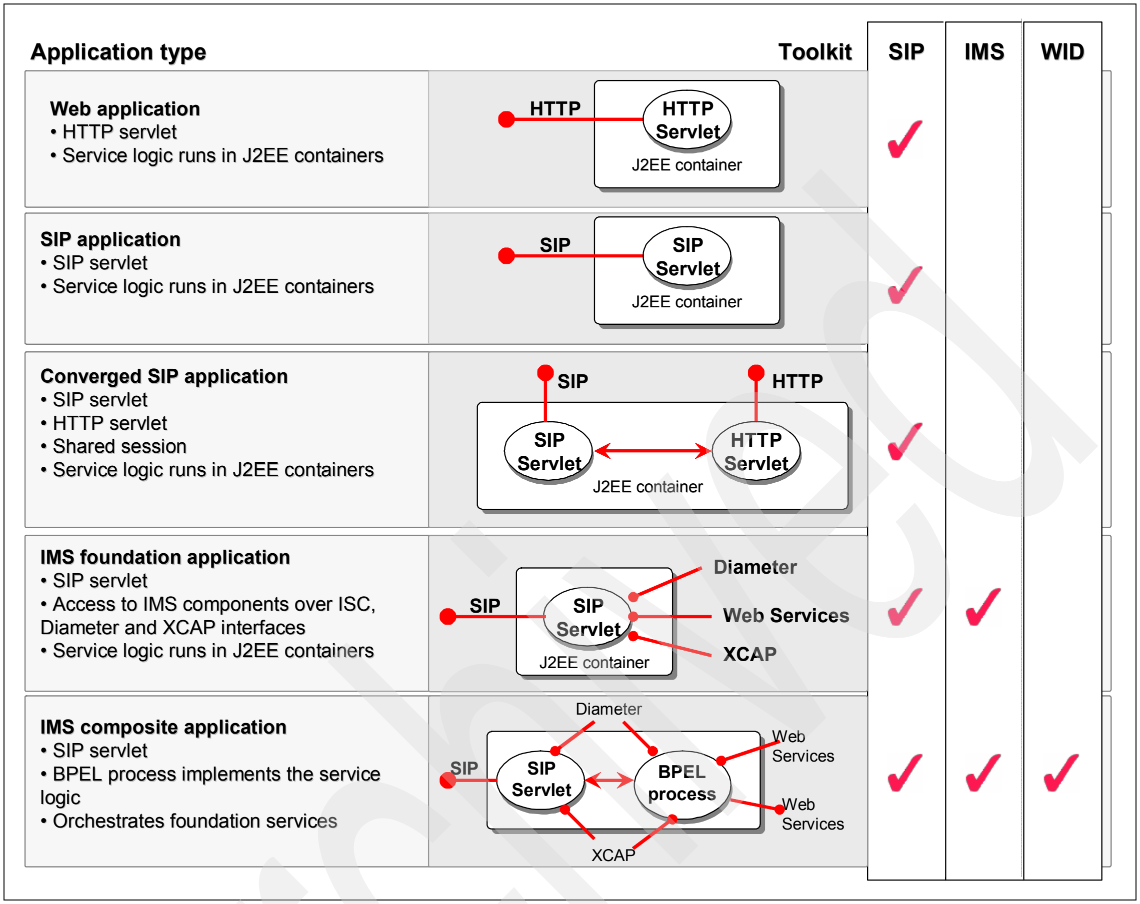


Figure 3-4 Application types and required toolkit

Note: USCE includes a number of toolkits for developing SIP and IMS applications. They include:

- IBM WebSphere Application Server Toolkit
- IBM WebSphere IMS Enablement Toolkit
- IBM Telecom Web Services Toolkit

These toolkits and their use in creating SIP and IMS applications is the focus of this redbook. In this chapter we introduce the toolkits and in the subsequent chapters we provide a more complete overview of each toolkit and use the toolkits to create sample SIP and IMS applications.

3.4 The SIP and IMS service creation environment

The IBM SIP and IMS service creation environment consist of a set of toolkits for developing SIP, converged and composite IMS applications, as well as servers that provide the runtime environment where these services run. Figure 3-6 on page 52 captures the key components of the SIP and IMS service creation environment.

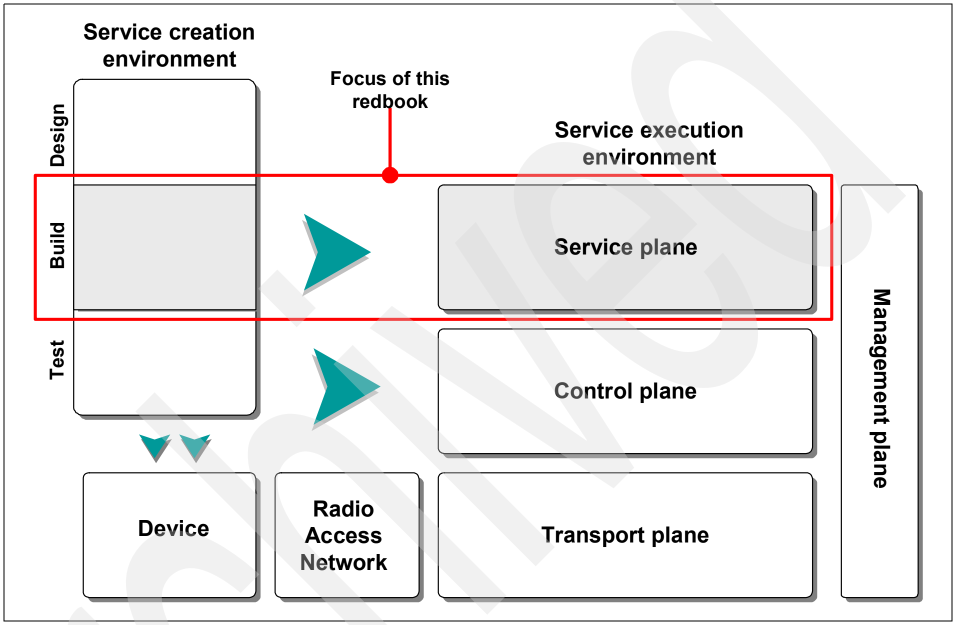


Figure 3-5 Service creation and execution in the IMS architecture

The key components of the IBM service creation environment include the toolkits:

- IBM WebSphere Application Server Toolkit which is used for creating SIP and SIP/HTTP converged applications
- IBM WebSphere IMS Enablement Toolkit which is used for creating IMS foundation applications.
- IBM Telecom Web Services Toolkit which is used for creating composite IMS applications.

The runtime environments are:

- WebSphere Application Server V6.1

  Provides runtime for SIP through its support for converged container for SIP and HTTP servlets.

- WebSphere Process Server

  Is a high-performance business engine that executes business processes securely, consistently, and with transactional integrity.

- WebSphere Enterprise Service Bus

  Provides Web Services connectivity, Java Message Service (JMS) messaging and service oriented integration.

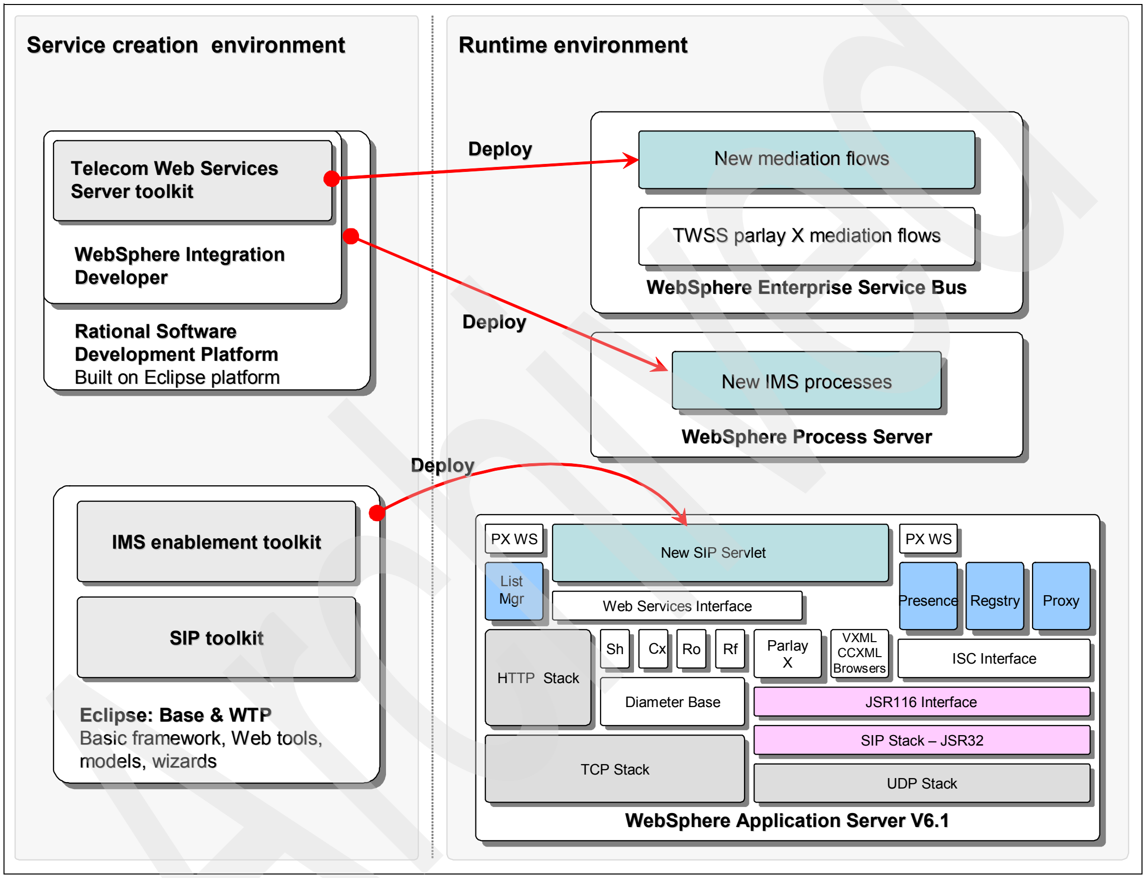


Figure 3-6 Overview of IBM service creation and execution environments

In the rest of this section, we introduce the toolkits and the runtime environment. An overview of each of the toolkits as well as the runtime environment is presented in the next five chapters.

Note: We used a number of third party tools which are not part of the IBM service creation environment for developing and testing SIP and IMS sample applications in this redbook. The tools include:

- SIPp

  SIPp is a free Open Source test tool / traffic generator for the SIP protocol available under GNU General Public License. The current version is 1.1rc5, but the one we have used when writing the redbook is 1.1rc4. It includes a few basic SipStone user agent scenarios (UAC and UAS). It can also reads custom XML scenario files containing SIP messages thus describing from very simple to complex call flows.

  http://sipp.sourceforge.net/

- Ethereal

  Ethereal is a network packet analyzer software available for both Windows® and Linux environments. It is Open Source software released under the GNU General Public License. It captures capture network packets flowing in and out of a selected network interface and displays them in real time with protocol dependent information. It provides filtering capabilities and supports SIP protocol.

  It can be found at:

  http://www.ethereal.com

- SIPxPhone

  SIPXphone is a fully functional SIP soft phone that runs on Microsoft® Windows and Linux. The phone client supports multiple simultaneous calls, hold, mute, client-mixed conferencing, consultative transfer, multiple line appearances, authentication, and an extensible Java-based application environment.

  sipXphone is developed under open source and hosted as part of the sipX line of projects available from SIPfoundry. It is licensed under LGPL. For more information about open source licensing or SIPfoundry, see:

  http://www.opensource.org or http://www.sipfoundry.org

3.4.1 IBM WebSphere Application Server Toolkit

The IBM WebSphere Application Server Toolkit (AST) is available with WebSphere Application Server (WAS) Version 6.1. You can use it to create, test and deploy applications that run within the WebSphere Application Server Version 6.1.

The AST is built using the Eclipse Web Tools Platform V1.0.2. All tools in the AST are integrated into the Eclipse workbench. The tools include wizards which are used to generate the set of files for Java, Java 2 Platform, Enterprise Edition (J2EE), Enterprise JavaBean (EJB™), SIP and Portlet projects, and editors which provide code assist and validation to improve productivity. AST is fully integration with WebSphere Application Server, so applications developed with AST can be deployed on WebSphere Application Server V6.1 for testing and execution.

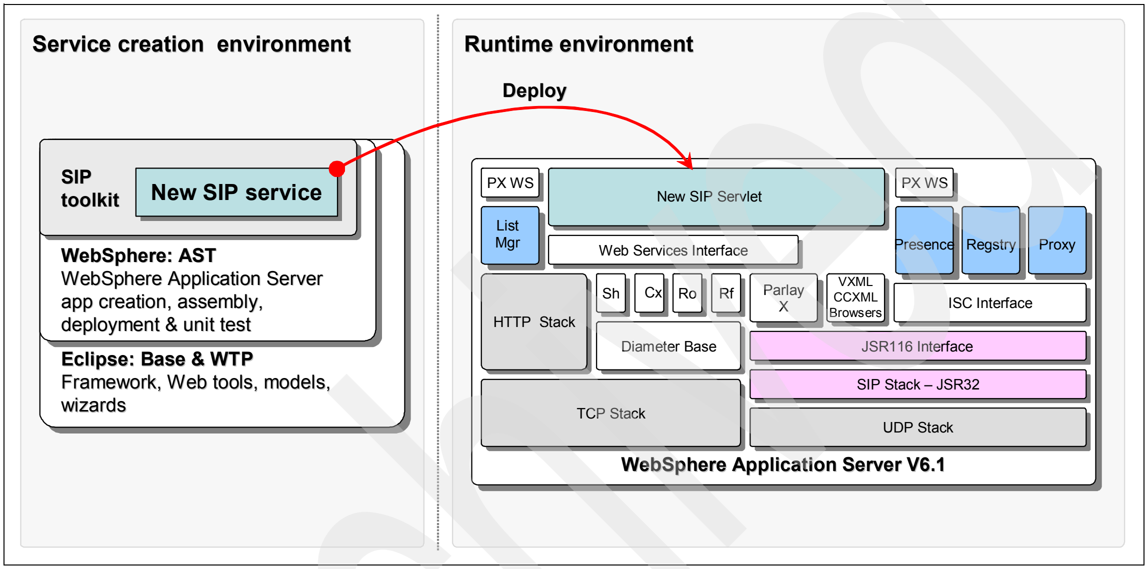


Figure 3-7 SIP application development and execution environment

Using AST, you are able to do the following:

- Develop applications ranging from Web to J2EE applications

  WebSphere AST supports a variety of technologies such as J2EE 1.4, Enterprise JavaBeans™ 2.1, Web Services, XML, and portlets.

- Program in different markup and scripting languages

  You can edit various markup languages such as HTML and XML, as well as JavaScript™.

- Manage source code

  The Eclipse workbench supports use of source code control system to manage, share and synchronize resources. It can be configured to work with CVS, Clearcase, or other source control systems.

- Deploy applications for unit test

You can build, package, and deploy applications for testing and debugging.

- Perform functional test

  You can deploy applications for function testing in a test server environment.

- Deploy for production

  Finally you can deploy the completed application onto a production server.

A more detailed overview of the IBM WebSphere Application Server Toolkit is presented in Chapter 4, “IBM WebSphere Application Server Toolkit” on page 63.

3.4.2 IBM IMS Enablement Toolkit

IMS EnablementToolkit adds additional features to the IBM WebSphere Application Server Toolkit to enable access to core IMS enablers such as presence and group list from the SIP servlet. The IMS enablement plug-in enhances the Java 2 Platform, Enterprise Edition (J2EE) development perspective to enable the creation of foundation level IMS applications.

The enhanced J2EE perspective includes:

- Service creation with new SIP project and servlets wizard to accelerate development of JSR 116 SIP Servlets.
- Support for the SIP archive (SAR) archive format and a wizard for editing SIP deployment descriptors, just like other J2EE components.
- Compilation with automatic inclusion of SIP Application Server (SIP A/S) libraries to decrease risk of errors.
- Packaging of J2EE and SIP components to accelerate deployment to runtime servers.
- SIP sample services gallery
- IMS specific help plug-in.

Foundation level applications make use of enablers such as Short Message Service (SMS) and location services as well as IMS enablers (for example presence) to provide encapsulated end-user services. They provide simple interfaces which enable reuse, making it possible to compose foundation level services to build more feature rich IMS composite applications.

The IBM WebSphere Platform for Telecom contains the following IMS enablers and access gateways:

- IBM WebSphere Presence Server

  IMS-compliant server that collects, manages, and distributes real-time information regarding the access, availability, and willingness to communicate of users.

- IBM WebSphere Group List Server component

  The group list server enable users and administrators to create and manage network-based groups. It maintains access lists, permissions, and other service-specific properties associated with those groups and group members across applications.

- IBM WebSphere IMS Connector

  WebSphere IMS Connector adds IMS-specific interfaces to the industry-leading WebSphere Application Server V6.1 platform to deliver for a fully IMS standards-compliant SIP application server.

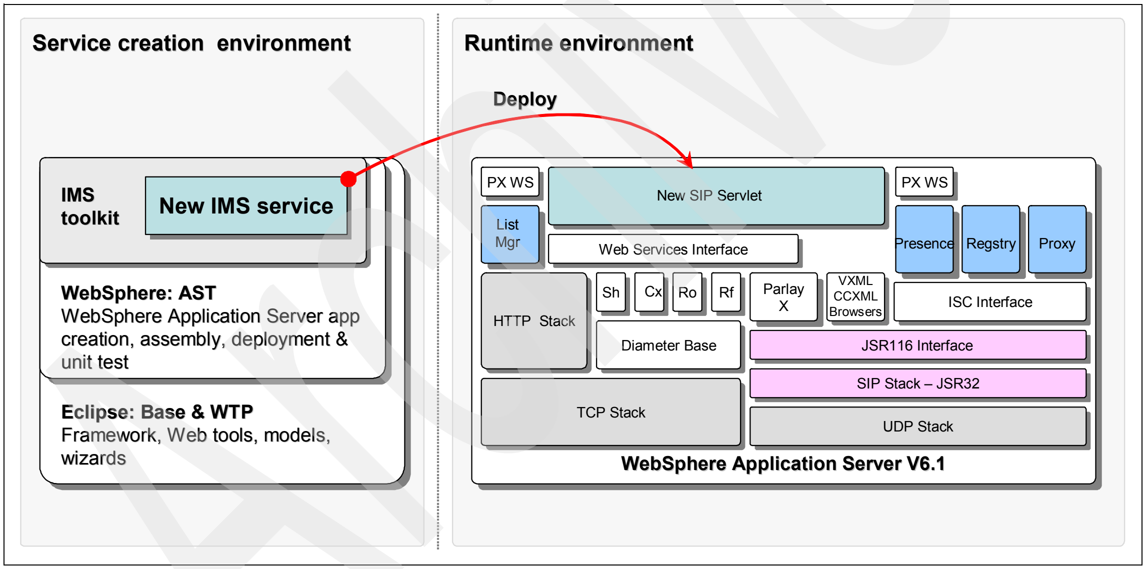


Figure 3-8 The service creation and execution environment for IMS foundation services

In Chapter 5, "IBM IMS Enablement Toolkit" on page 89 we present more detailed overview of the IMS Enablement Toolkit.

3.4.3 IBM Telecom Web Services Toolkit

IMS composite applications consist of foundation level applications and service enablers that are composed to create yet higher level services using the IBM Telecom Web Services Toolkit and the IBM WebSphere Integration Developer.

A key requirement for foundation services as well as service enablers is the adherence to the principles of service-oriented architecture (SOA). In other words, foundation services and service enablers, act as SOA service implementations by exposing well defined interfaces (the SOA service) and avoiding any implicit dependencies on other components. The services and enablers are choreographed using BPEL resulting in new composite services. Note that the new composite services are themselves SOA services and hence can be used to create yet higher level services.

To develop IMS composite application you need to install the IBM Telecom Web Services Toolkit which is a BPEL process component to the base IMS service creation environment. You also need the IBM WebSphere Integration Developer which is an integrated and powerful development platform for BPEL process development.

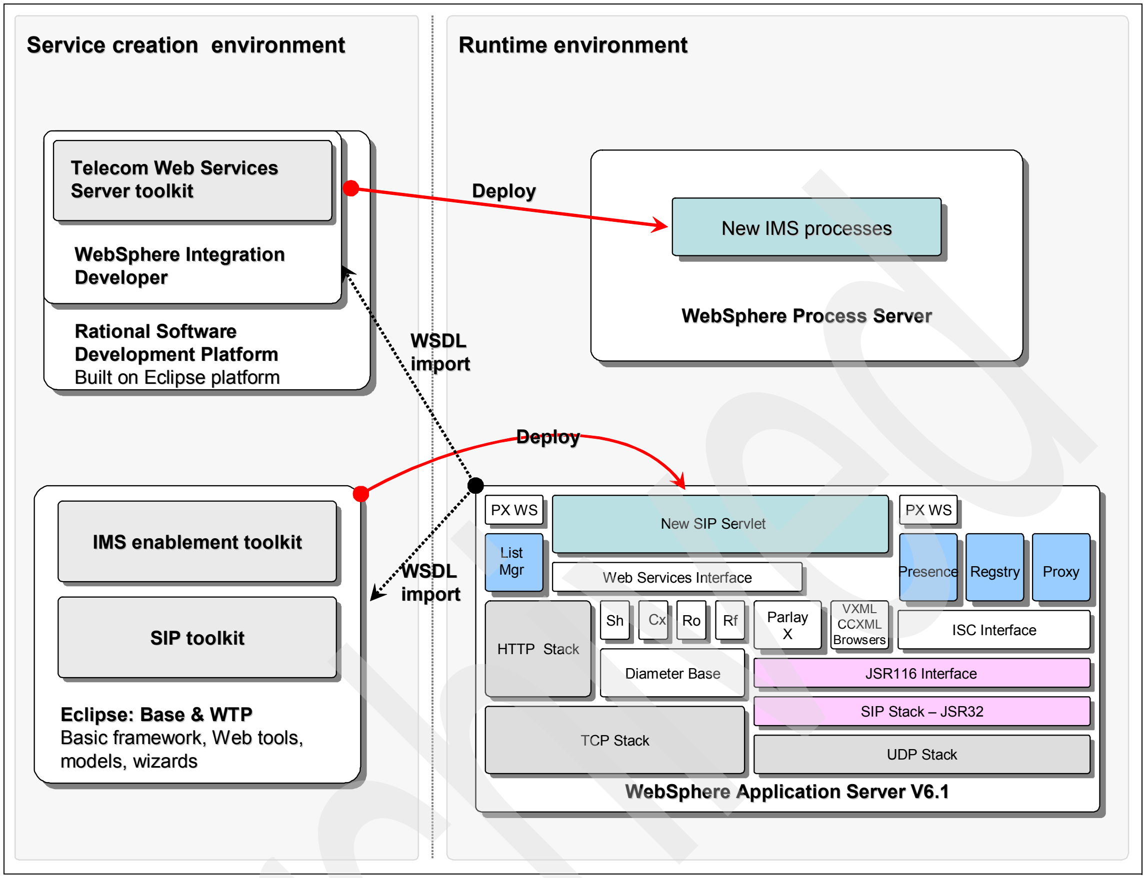


Figure 3-9 IMS development and execution environment for composite services

3.4.4 IMS Enablement Toolkit

The IMS Enablement Toolkit 6.1 functions in Eclipse 3.1 and Web Tools Platform (WTP) 1.0.2. It enables the development of IMS applications using Standards-compliant IMS extensions to IETF SIP through the ISC (IMS Service Control) interface.

The IMS Enablement Toolkit 6.1 includes the following components:

- Diameter Resources
  - Rf Interface libraries - Rf accounting Web Services provides an IMS Application Server application with a Diameter messaging interface to enable the application to send offline accounting messages to accounting or billing servers

  - Sh interface libraries - Sh subscriber profile Web Services are used to retrieve and update user profile data from the Home Subscriber Server (HSS)
  - Corresponding Web Services Description Languages (WSDLs) of Rf and Sh services to enable interaction between IMS application server and charging and subscriber profile servers through Web Services. It also includes a notification WSDL to implement to receive notifications that a specified subscriber has been changed
  - Rf and Sh test clients
- Presence Resources
  - Authorization APIs library - allow for the development of customized permission policies. With these APIs it is possible to develop pluggable applications that provide authorization information and allow or disallow subscription to a *presentity*.
- Parlay X 2.1 Resources
  - WSDLs for Parlay X 2.1 Web Services (they include Third Party Call, Call Notification, SMS, Payment and Terminal Status)
- An example of ISC SIP project

  This sample demonstrates the use of SIP Servlet API and ISC to implement a Back-To-Back User Agent (B2BUA) service. It shows the interaction between User Agents, SIP servers and Call Session Control Function (CSCF).

IBM WebSphere Integration Developer

The IBM WebSphere Integration Developer is a role-based development environment based on the Eclipse 3.0 platform. It can be used in conjunction with other Rational and WebSphere tools. Each user has a unique tools perspective based on their role (for example, J2EE developer, business analyst, or integration developer).

The IBM WebSphere Integration Developer supports both a top-down design approach to building integrated applications, where the implementation for one or more components does not already exist and is added later; as well as a bottom-up approach, where the components are already implemented and the developer assembles the components, creating a logical flow by connecting the components using a visual editor. It also provides a debugging and test environment which includes setting of points for real time monitoring and fine tuning for optimal performance.

Applications created using the WebSphere Integration Developer conform to a number of industry-wide standards. These include:

- J2EE Connector Architecture which is used for connectivity

- Java Message Service (JMS) is used for asynchronous messaging, and in cases where guaranteed delivery of data is required
- Simple Object Access Protocol (SOAP) is used for integrating Web Services
- Web Services Description Language (WSDL) for describing services
- Business Process Execution Language (BPEL) to define business processes

The standards-based interfaces and components provide an open and pluggable architecture which supports integration of components developed on other platforms such as .NET.

In addition to the business process and integration, WebSphere Integration Developer also provides support for development of mediation services.

> **Attention:** Mediation services in the context of the Enterprise Service Bus architecture is different and should not be confused with mediation services typically used to collect and transform charging records in BSS/OSS context.

Mediation services intercept and modify messages that are passed between existing services (providers) and clients (requesters) that want to use those services. Mediation modules can be deployed on the WebSphere Enterprise Service Bus or the WebSphere Process Server.

For example, mediation flows can be used to find services with specific characteristics that a requester is seeking and to resolve interface differences between requesters and providers. For complex interactions, mediation primitives can be linked sequentially. Typical mediations include:

- Transforming a message from the sending service to a format that the receiving service can process
- Conditionally routing a message to one or more target services based on the contents of the message
- Augmenting a message by adding data from a data source

Chapter 7, "IBM Telecom Web Services Server Toolkit" on page 145 and Chapter 6, "IBM WebSphere Integration Developer" on page 119 provide overviews of the IBM Telecom Web Services Toolkit and the IBM WebSphere Integration Developer, respectively.

3.5 The service execution environment

The service execution environment provides the platform with standardized interfaces for executing applications and performing authorized operations.

Figure 3-10 is an illustration of the IBM service execution environment which shows the modular service enablers the IBM WebSphere IP Multimedia Subsystem (IMS) Connector, IBM WebSphere Presence Server and IBM WebSphere Telecom Web Services Server.

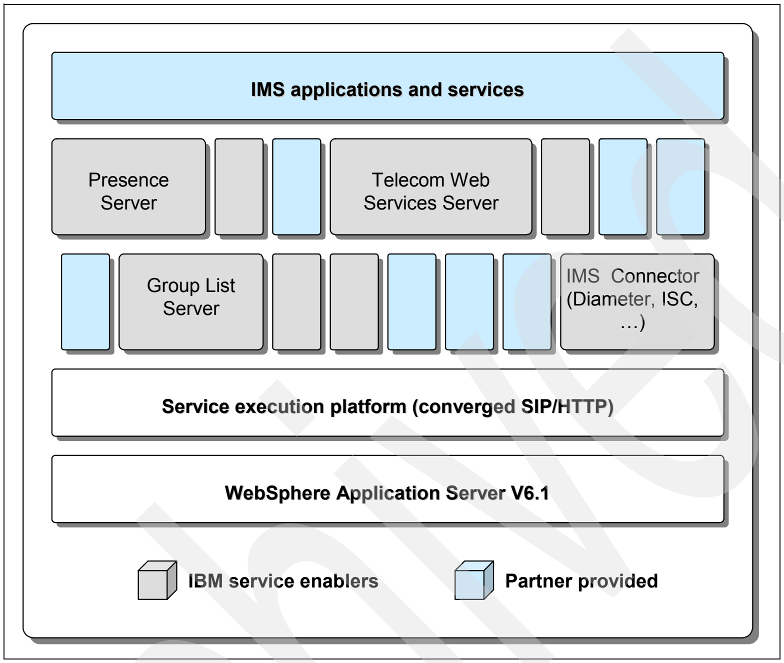


Figure 3-10 IBM IMS service execution environment

The environment is delivered on the WebSphere software platform, the proven, secure and flexible environment that delivers high volume and mission-critical infrastructure. The core execution environment is provided by WebSphere Application Server, which in this latest 6.1 release includes a SIP stack. The IBM service execution environment may also include the WebSphere Process Server which contains the WebSphere Enterprise Service Bus. The Process Server provides a high-performance business process engine that executes composite services securely, consistently, and with transactional integrity. A detailed overview of the service execution environment is described in Chapter 8, “Introduction to the IBM service execution environment” on page 159.

4

IBM WebSphere Application Server Toolkit

This chapter provides an introductory overview of the WebSphere Application Server Toolkit (AST) an Eclipse based integrated application development environment. It introduces the use of AST for developing, packaging and deploying SIP applications that run in the WebSphere Application Server V6.1.

This chapter contains the following:

- AST overview
- Developing SIP servlet application
- SIP servlet deployment
- Sample SIP services

4.1 AST overview

The WebSphere Application Server Toolkit (AST) is an application development environment for creating applications targeting the WebSphere Application Server. Using AST, you are able to create, test and deploy applications to the WebSphere Application Server. The tools in AST are integrated into a workbench that is based on Eclipse technology and the Web Tools Platform.

AST is just one of the tools in the IBM hierarchy of integrated application development environments. Figure 4-1 shows the illustration of the hierarchy.

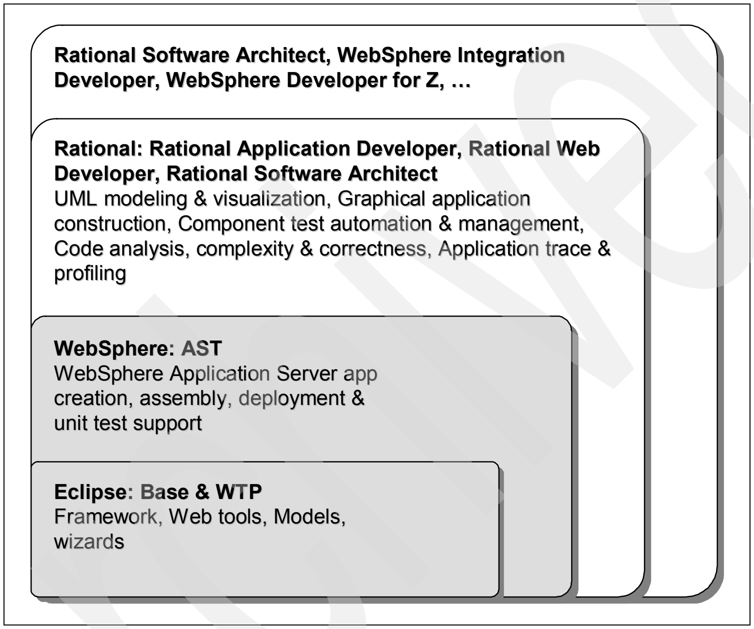


Figure 4-1 IBM tools portfolio hierarchy

With the release of WebSphere Application Server V6.1, AST 6.1 has transitioned from its focus as an application deployment tool to an application development tool. In addition to expanded functional scope of WebSphere tools, AST now include the following new features:

- Inclusion of Eclipse Web Tools Platform V1.0.2
- Jython development tools
- Tools support for developing JSR 168 portlets
- Improved server tools for publishing and testing applications targeted for WebSphere Application Server V6.1

- And of interest for the topic of this book, tools support for developing JSR 116 SIP servlet applications

The workbench incudes J2EE perspective which now allow for the creation of Web services from JavaBeans, Enterprise JavaBeans and WSDL files. Previously, you could only assemble and deploy Enterprise JavaBeans, as well as modify their deployment descriptor. Now you can create, modify and deploy EJBs, do bottom-up mapping for container managed persistence (CMP) and support back-end database for generating EJB deployment code.

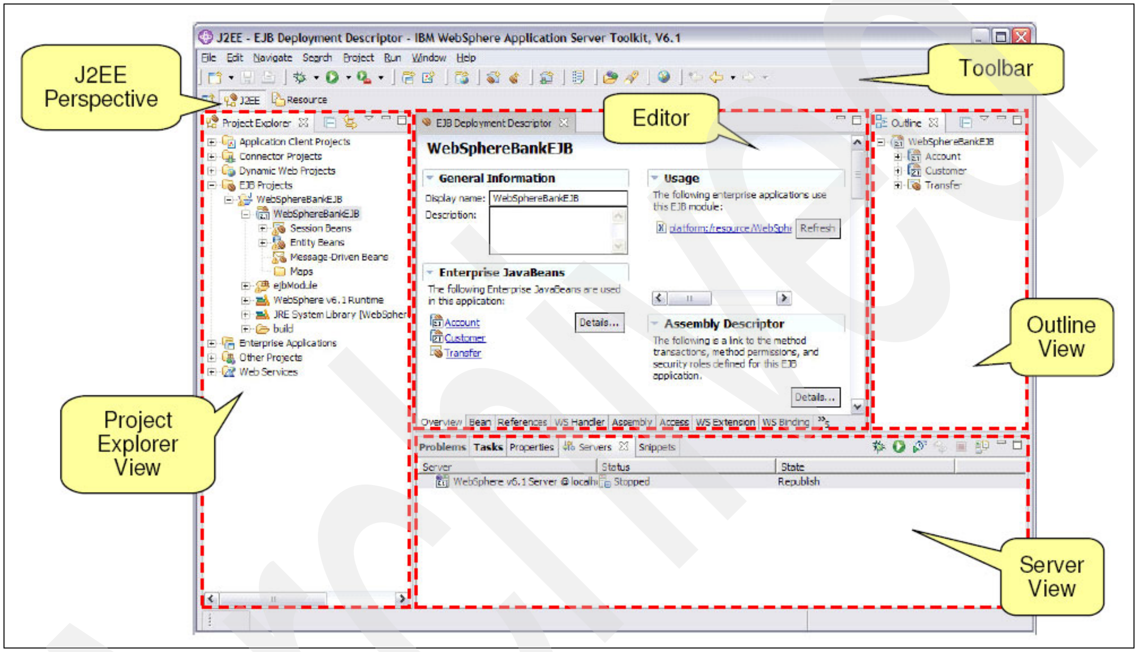


Figure 4-2 The AST workbench

AST V6.1 supports a number of SIP application development tools that are available through the J2EE perspective. They include the following:

- SIP and converged SIP/HTTP projects

  AST introduced two new types of projects for creating SIP only and converged SIP/HTTP applications.

- SIP servlet development (JSR 116)

  In addition to HTTP servlet development, AST V6.1 includes the capability to develop SIP servlet based on the JSR 116 specification.

- SIP deployment descriptor editor

  The editor is used for configuring and packaging SIP applications. SIP applications are packaged in a new construct called a SIP Application Resource (SAR). It can encapsulate SIP Servlets and traditional HttpServlets and both are fully supported by AST V6.1.

  Using the SIP deployment descriptor editor you are able to configure and specify parameters for different servlets in your application. For example, you can use the deployment descriptor editor to add servlet mappings so SIP messages are routed to appropriate servlets for processing.

- Import/export of SAR packages

  Unlike other Web and J2EE applications, SIP applications cannot be directly deployed on the application server from inside AST. You must first use the import/export tool to export the SIP application as either a standalone SAR or as part of a Enterprise ARchive (EAR) package. Only then can you install the package to the application server using the administrative console.

> **Note:** AST V6.1 main and enhanced features
>
> - Server tools for WebSphere Application Server, such as debugging and unit testing support.
> - Support for WebSphere Application Server-specific extensions, such as SIP and Jython tools.
> - Graphical editors for WebSphere Application Server property files and deployment descriptors.
>
> Information center for AST V6.1 is can be accessed at the following URL:
>
> http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.welcome.ast.doc/topics/astoverview.html

4.2 Developing SIP servlet application

AST in WebSphere Application Server 6.1 adds the capability to develop JSR 116 Servlets, also called Siplets. You can develop and test SIP applications using the same methodology that is used to test and deploy J2EE applications. A Session Initiation Protocol (SIP) application is a group of servlets, resources and source files that can be managed as a single unit. You can develop SIP only or converged SIP and HTTP servlets using the AST

4.2.1 SIP only applications

To develop a new SIP application, you start by creating a new SIP project. And for that, you use the **New SIP Project** wizard. The wizard creates a SIP project similar in layout and content to a Dynamic Web Project. It includes additional categories on the project creation menu for **SIP Project**, **Converged Project** and **SIP Servlet**.

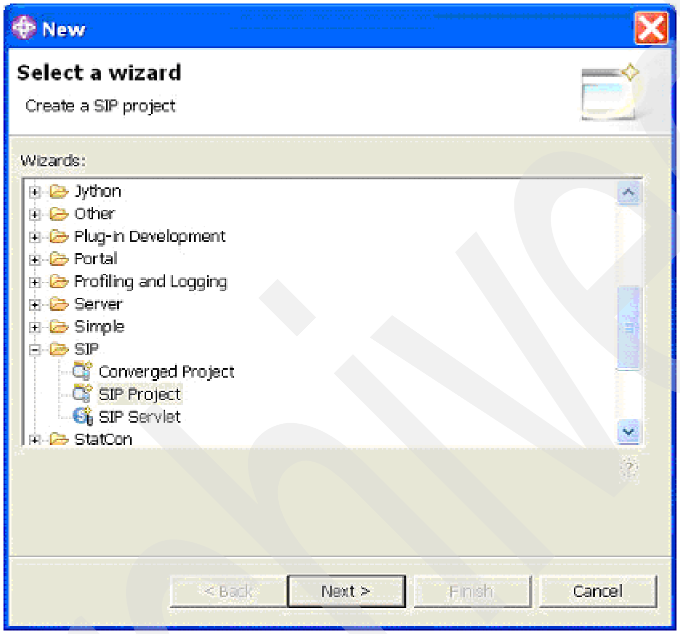


Figure 4-3 Creating a SIP project

Next, you create SIP Servlets which provide the application functions. You create a SIP servlet by defining the servlet class name, location, package and super class. This is identical to the HTTP servlet wizard's entries except that the superclass is pre-filled with the SIP Servlet API superclass, `javax.servlet.sip.SipServlet`.

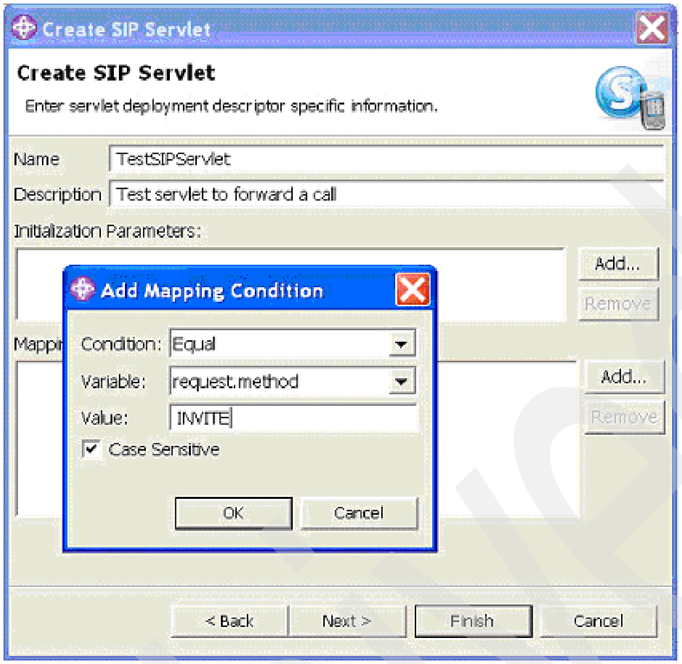


Figure 4-4 Creating A SIP Servlet

The deployment descriptor information is similar to that in the Servlet wizard, and you use it to enter the SIP servlet deployment descriptor information in the unique SIP mapping section.

> **Note:** More information about the definition of the deployment descriptor is provided in 4.2.3, "SIP servlet deployment" on page 71.

The last page in the wizard is used to defining interfaces and methods. Stubs are automatically created in the servlet class, you implement the servlet by developing the code for the stubs.

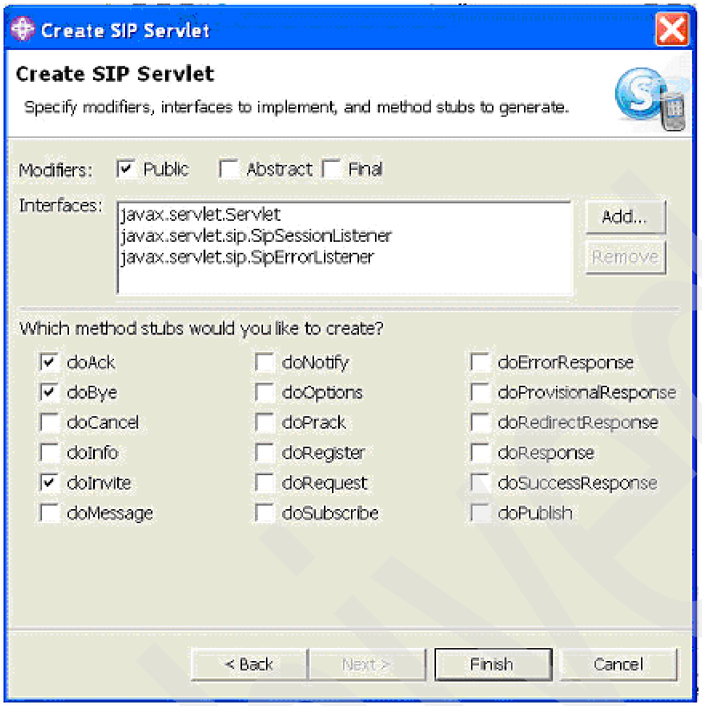


Figure 4-5 Choosing methods to implement in a SIP servlet

Note: A sample application showing how to develop SIP servlet applications using AST is provided in Part 3, "SIP applications" on page 203.

4.2.2 Converged SIP/HTTP applications

AST also supports creation of converged Session Initiation Protocol (SIP) and Hypertext Transfer Protocol (HTTP) applications. The converged project is a Dynamic Web project with SIP content and contains SIP deployment descriptor file, `sip.xml`, as well as Web deployment descriptor file, `web.xml`, located in the `WebContent\WEB-INF` folder.

Note: The current version of IBM SIP container supports servlet Version 2.3. This implies that when creating a Converged Project the Dynamic Web Module Version 2.3 will be automatically chosen.

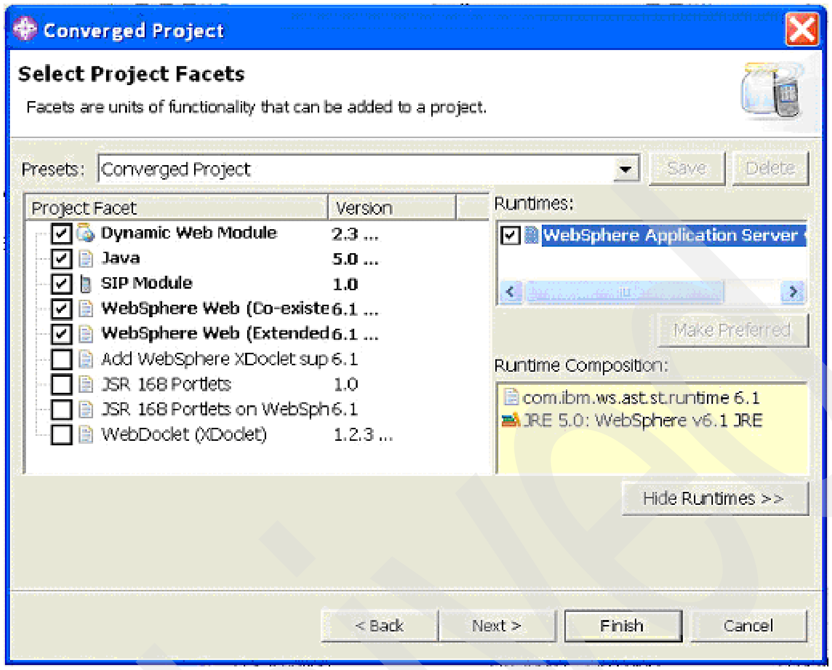


Figure 4-6 Dynamic Web Module version in a Converged Project

> **Note:** If you start a Web Application with HTTP Servlets 2.4, adding SIP content will be disabled.

If you are working with an existing Dynamic Web Project with Dynamic Web Module 2.3 that doesn't have SIP content in it, you can add SIP Content to it by selecting the folder, open the property editor for the project, and then select **Project Facets.** Click **Add/Remove Project Facets**, this will show a window as in Figure 4-7 on page 71 where you can select **SIP Module 1.0**.

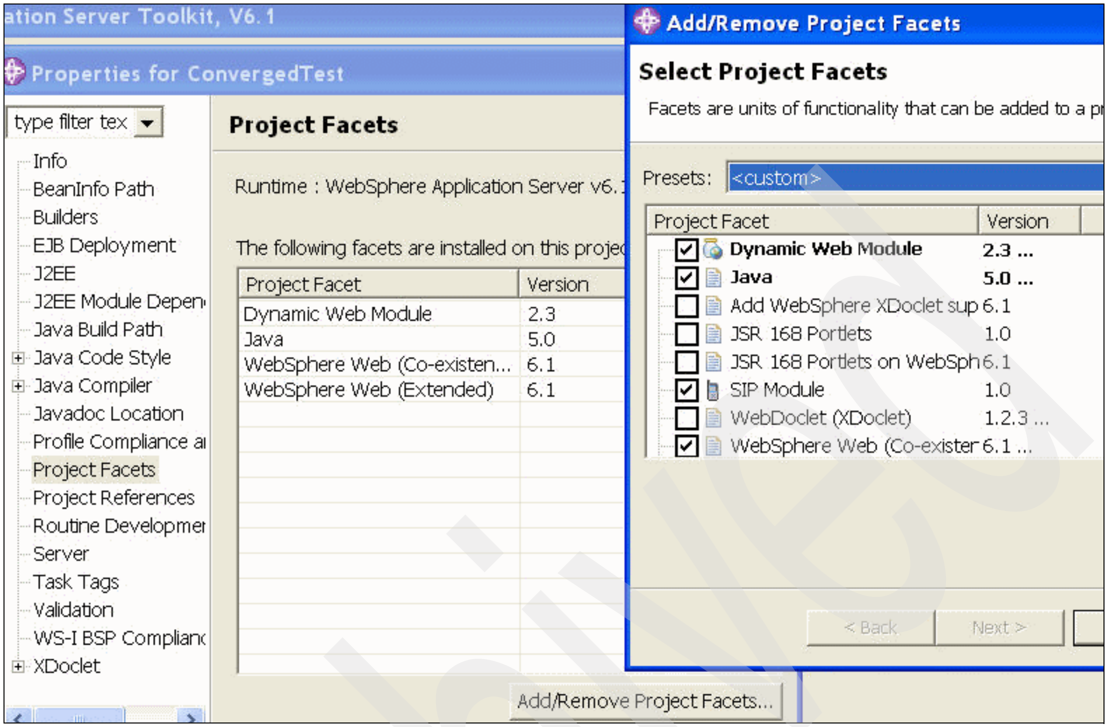


Figure 4-7 Adding a SIP Module to a Converged Project

If the existing Web Project already has HTTP Servlets, as per the SIP specification, the distributable, display-name, icons tag from web.xml will be copied to the newly created sip.xml. This avoids having to manually check the consistency between sip.xml and web.xml. Once the SIP deployment descriptor is created, it is possible to add SIP Servlets from the deployment descriptor editor.

4.2.3 SIP servlet deployment

The deployable SIP application is a SAR file created by the export wizard. It is a basic Java archive with a ".sar" file extension. In addition to project resources, the SAR file includes the SIP deployment descriptor file.

The SAR export and import wizards are extensions of the Web archive (WAR) export and import wizards. The SAR export wizard allow you to specify the SIP project to export, and whether to include source files or not.

In the case of Converged SIP/HTTP application, the SAR export operation has additional validations during export process to ensure compliance with SIP specification. In particular, the initialization parameters for SIP and HTTP are merged in the ServletContext. The converged application may be exported to a WAR file, however the above check will not be performed.

The import wizard is essentially the reverse of the export wizard. With the import wizard, you are able to import a previously exported SAR file into your workspace. If the SAR file is a SIP only application, then a corresponding SIP Project is created. If on the otherhand the SAR file is a converged HTTP/SIP application, a project with HTTP and SIP content will be created.

Deployment descriptor

The SIP Deployment Descriptor (DD) is used to describe how a SIP application should be deployed. XML is used for the syntax of the Deployment Descriptor file. The information is stored in `.xml` file, for a SIP application the file is `sip.xml` and for HTTP applications the file is `web.xml`. The deployment descriptor information is used to build the SAR file when you export the SIP application.

AST provides a SIP Deployment Descriptor editor for creating and maintaining the `sip.xml` descriptor file, and the Web Deployment Descriptor editor for `web.xml`

Web deployment descriptor editor

The Web deployment descriptor editor lets you specify deployment information for modules created in the Web development environment. The information appears in the `web.xml` file. You should use the Web deployment descriptor to set deployment descriptor attributes and not to manipulate Web resource content.

The `web.xml` file for a Web project provides information necessary for deploying a Web application module. It is used in building a WAR file from a project. The web.xml file is automatically created in WEB-INF under the project's Web content folder.

The Web deployment descriptor editor is dynamic and includes many tabbed pages (views) that represent various properties and settings in the deployment descriptor. For example, you can click the Servlets tab to display the servlets page, where you can add or remove servlets and JSPs that are used in the Web application.

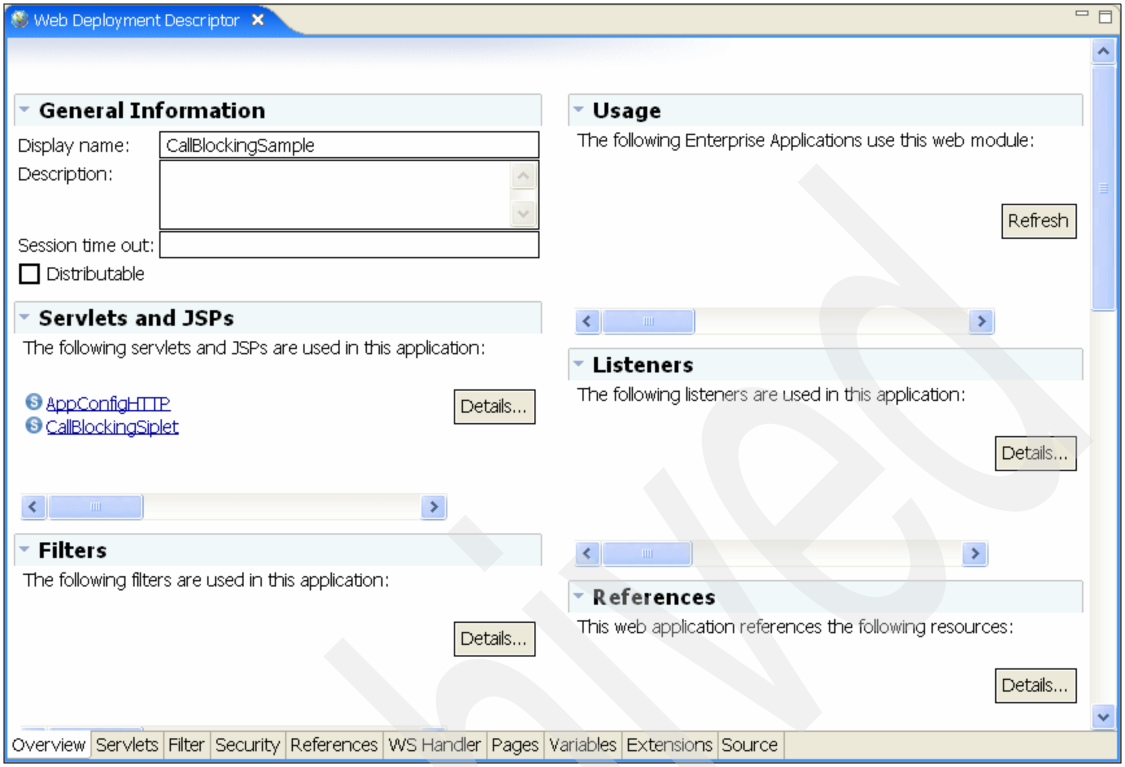


Figure 4-8 Web deployment descriptor editor

SIP deployment descriptor editor

The SIP deployment descriptor editor includes multiple tabbed pages (Overview, Servlet, Security, Variables, References and the Source page), each of which you can just view to get a summary of its contents or you can add, remove, or change the contents.

- Overview page

  The Overview page extends the Web Deployment Descriptor Overview Page. Some pages and sections that are not relevant to SIP have been removed while additional sections relevant to SIP are added for example the Login section.

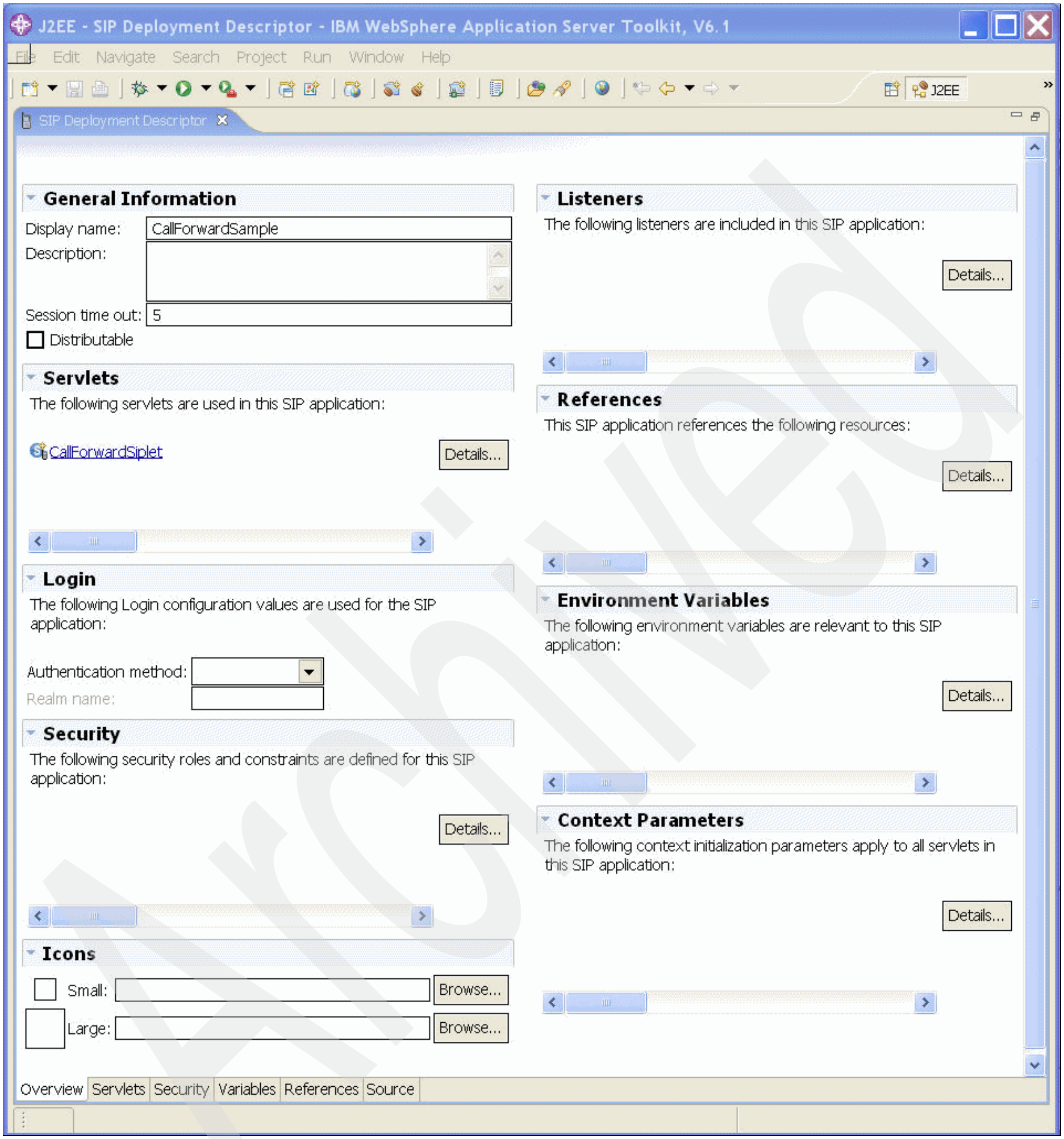


Figure 4-9 SIP Servlet Descriptor Editor Overview page

The sections of the Overview Page include:

– General Information

  Defines general SIP project settings such as Display name, Description, Session time out, and whether the application is distributable.

– Servlets

  Lists the servlets defined for the application. Clicking the 'Details' button shows the Servlets page.

– Login

  Defines the Login configuration. The options for Authentication method are: BASIC, DIGEST, and CLIENT_CERT.

– Security

  Lists the security roles and constraints defined for the application. Clicking the 'Details' button shows the Security page.

– Icons

  Defines the applications small and large icons. The 'Browse' buttons open the appropriate dialogs to allow the user to select images already defined in the project.

– Listeners

  Lists the listeners defined for the application. Clicking the 'Details' button shows the Variables page.

– References

  Lists the references currently defined for this application. Clicking the 'Details' button shows the References page.

– Environment Variables

  Lists the environment variables defined for the application. Clicking the 'Details' button shows the Variables page.

– Context Parameters

  Lists the context parameters defined for the application. Clicking the 'Details' button shows the Variables page.

- Servlet page

  The Servlet page enable you to add, modify and remove servlets in the project. It extends the Servlets page from the Web Deployment Descriptor. Most sections are the same as the Web deployment descriptor, with the following exceptions. The URL Mapping section is replaced with a Mapping section specific for SIP. Also, the WebSphere Programming Model Extensions and WebSphere Extension sections are removed.

- Security page

  The Security page enable you to manage security roles and constraints for the project. This page extends the Security page of the Web deployment descriptor editor. It replaces the Web Resource Collection section with the SIP Resource Collection sections.

  – Security role

    This subsection allow for viewing or addition of security roles.

  – Security constraints

    This subsection allows viewing or addition of constraints. Clicking the Add button displays the entry window where you enter the SIP servlet, select the SIP methods and associated constraints.

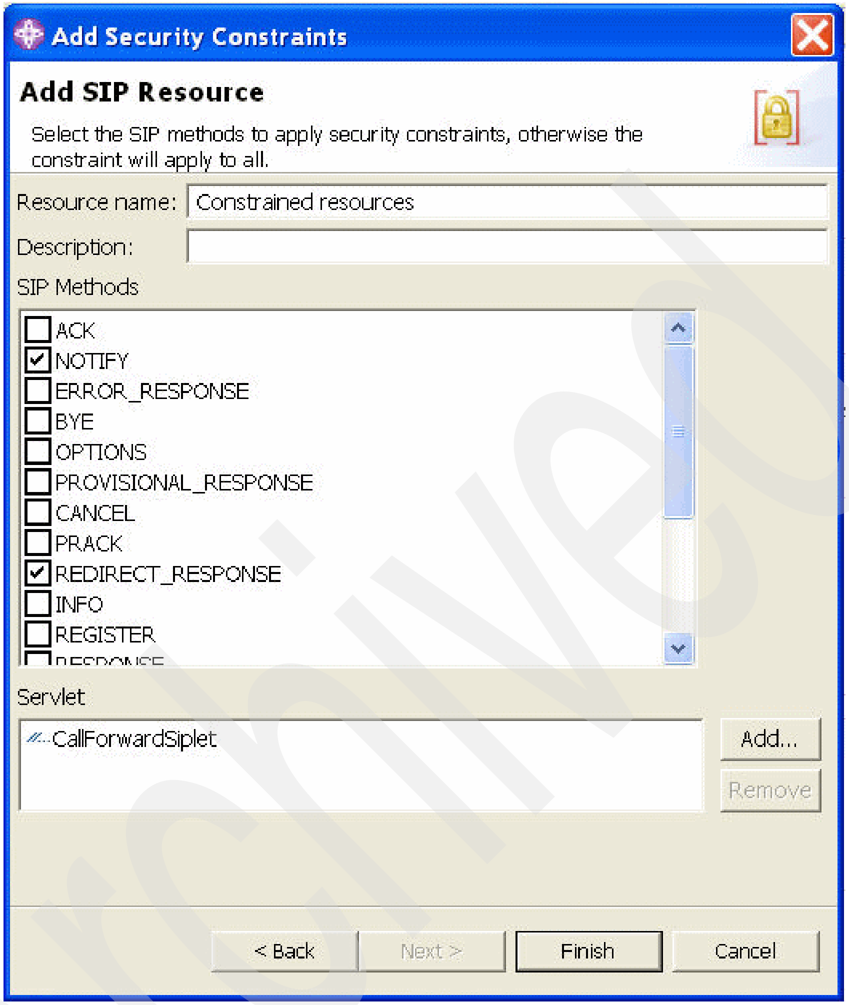


Figure 4-10 Adding Security Constraints to a SIP Servlets

The result is displayed on the Security page as illustrated in Figure 4-11.

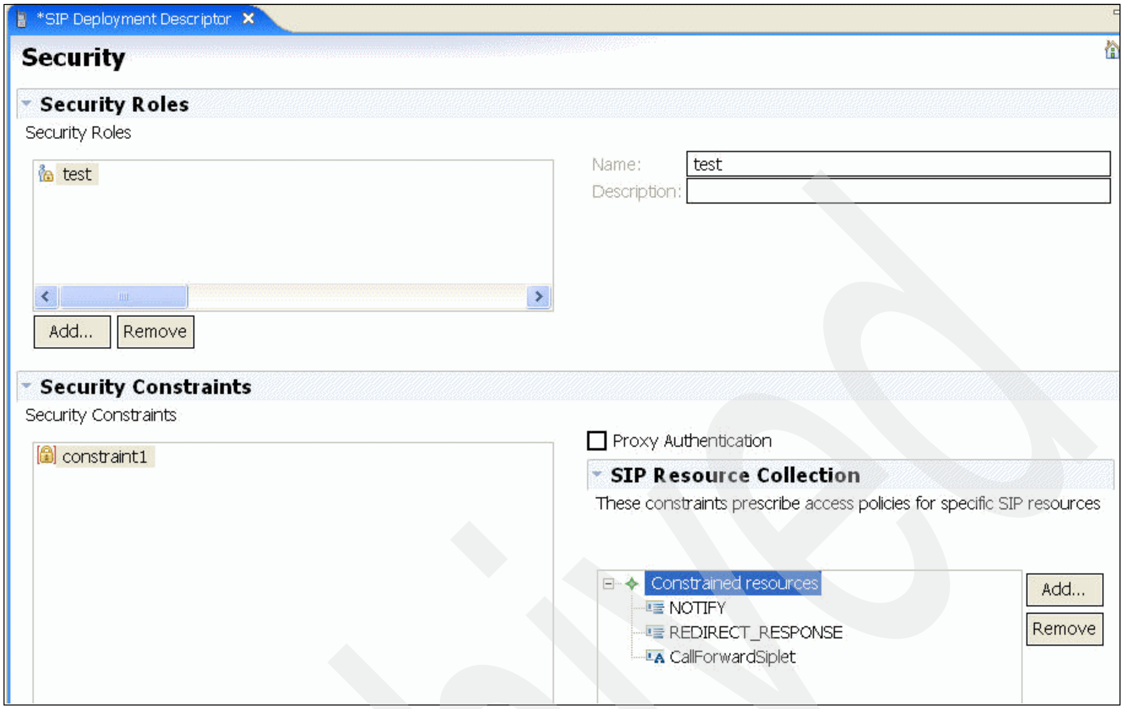


Figure 4-11 SIP Deployment Descriptor Security page

– Authorized role

The Authorized roles sub-section enable you to add roles for the selected security Constraint. This is the standard section of the Web tools. It allows definition of roles authorized to access the SIP resource collections of the predefined security constraint.

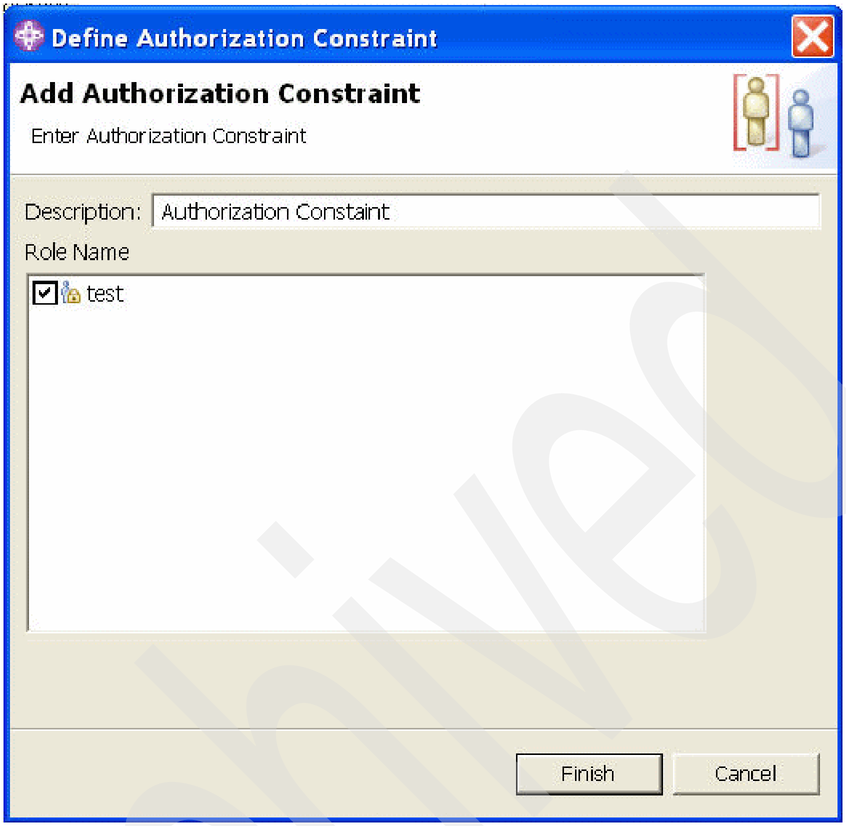


Figure 4-12 Adding Authorization Constraints

The result is displayed on the Security page, showing all fields filled up as illustrated in Figure 4-13 on page 80.

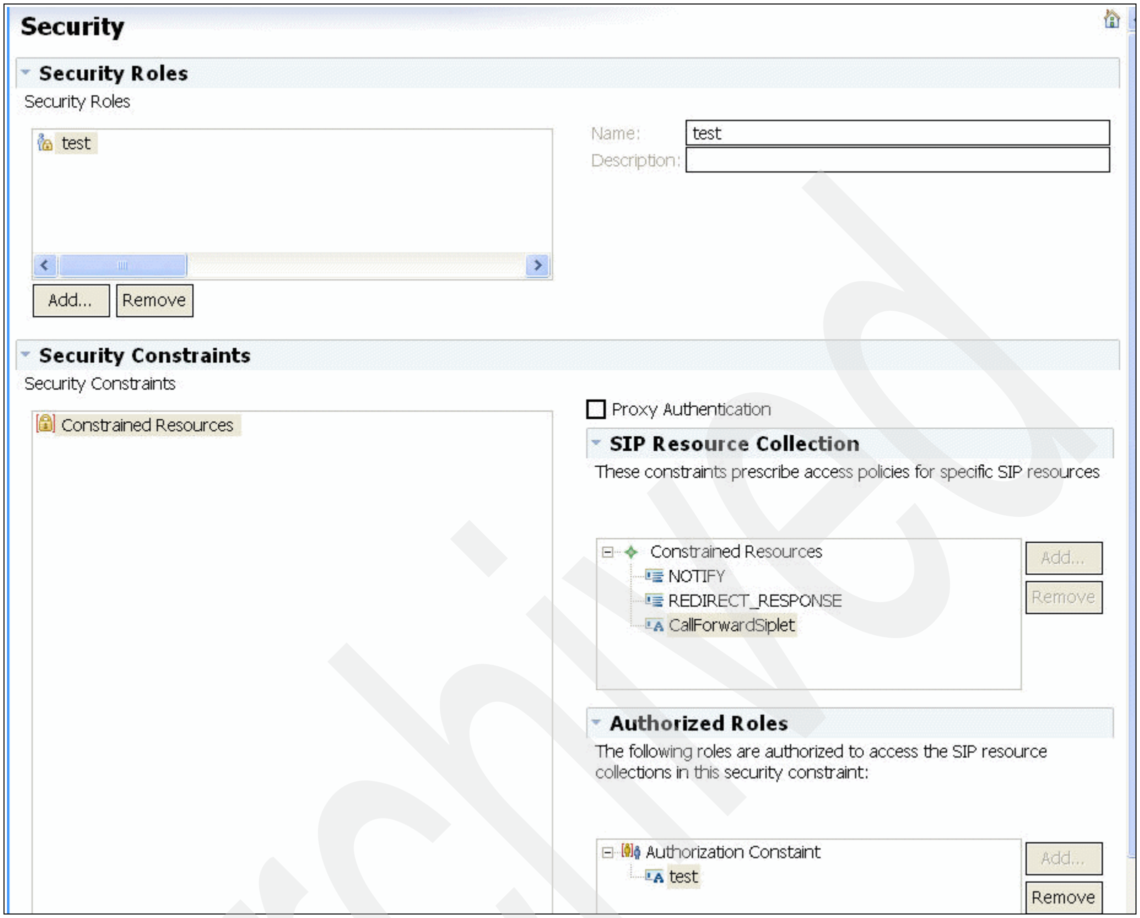


Figure 4-13 SIP Deployment Descriptor with defined Security Constraints

- Variable page

  The Variable page enable you to manage the list of listeners, context parameters and environment variables in the project.

  You can add or remove the following from the variables entry window:

  – Listeners

    This defines the <listener> elements in the sip.xml.

  – Context Parameters

    This defines the <context-param> elements in the sip.xml.

  – Environment Variables

    This defines the <env-entry> elements in the sip.xml.

Figure 4-14 shows an example of defining Life-cycle listeners.

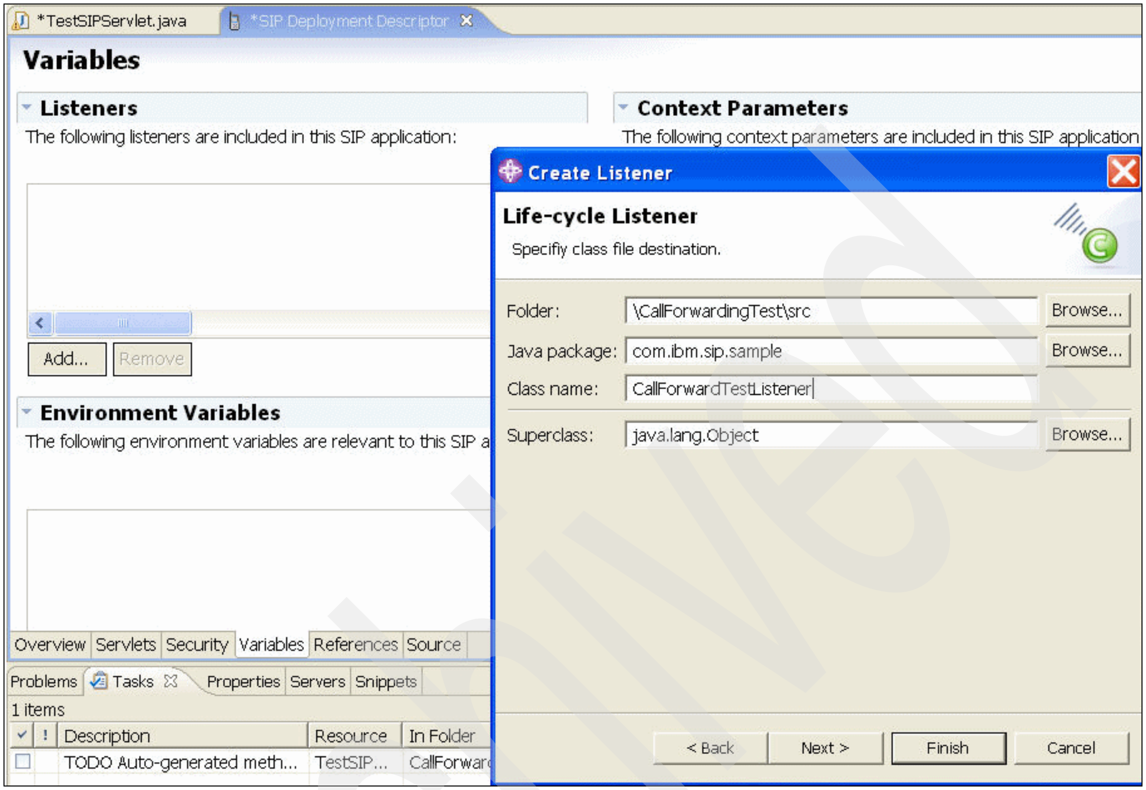


Figure 4-14 SIP Deployment Descriptor Life-cycle Listener definition

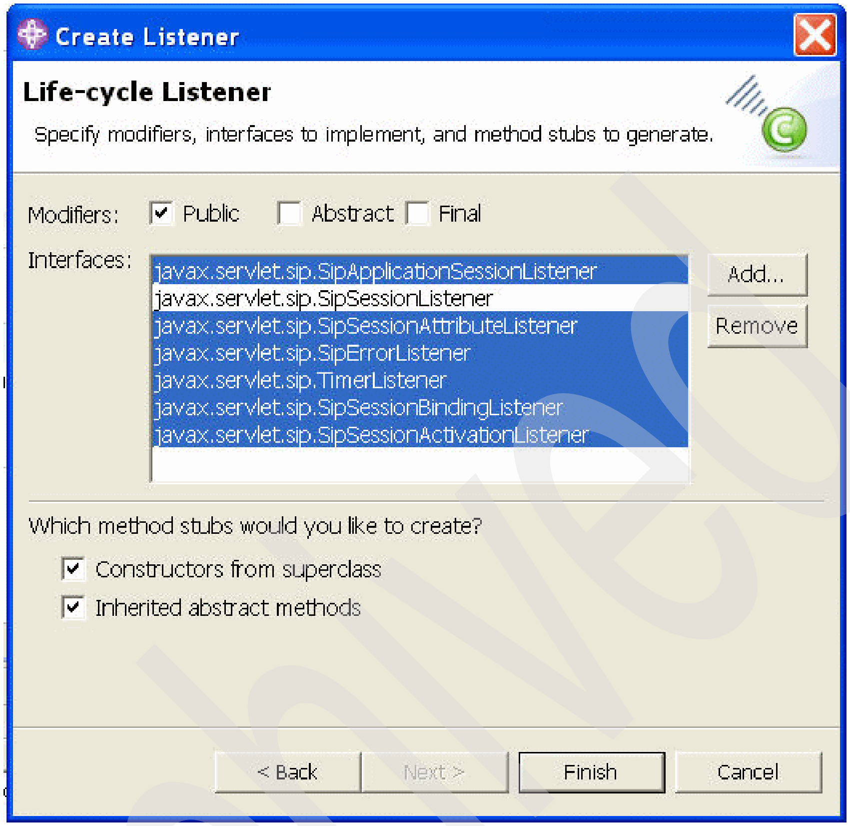


Figure 4-15 Specifying Interfaces for Life-cycle Listeners

- References page

  The References page enable you to manage the project's resource references. The following types of references are supported: local EJB reference, remote EJB reference, resource reference, and resource environment reference.

- Source page

  And finally the Source page displays the generated `sip.xml` source code.

4.2.4 Sample SIP services

AST contains several Samples that demonstrate particular Session Initiation Protocol (SIP) capabilities. To access these Samples:

1. From the **Help** menu, click **Samples Gallery**.
2. Click **Technology Samples > SIP**.

Three samples are included and each has their own page, which includes setup instructions as well as a wizard that will import the sample into the workspace

The samples that are included with the toolkit are:

- Call Blocking sample
- Call Forwarding sample
- Third Party Call Control

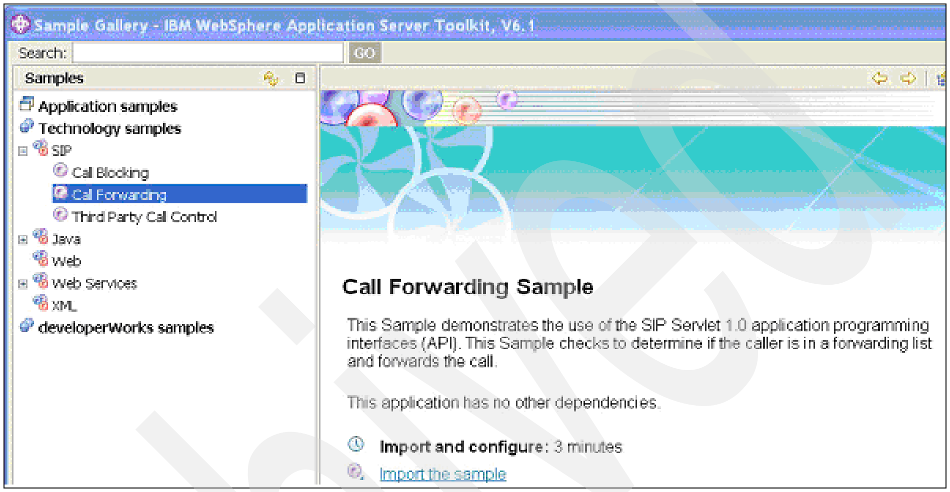


Figure 4-16 Loading Samples into AST

Call blocking sample

This sample demonstrates the use of SIP Servlet 1.0 APIs using ISC components. The sample checks a simple list (access list file) to determine if the caller is valid. The list is stored in a file which is part of the deployment package. If the caller is blocked for a particular callee, then the call is rejected. If accepted, the call is sent to proxy to send an invite to the callee. This sample can be customized by adding to this forward list and rerunning the sample.

Once loaded in AST, you can see, in the src\com.ibm.siptools.samples folder, three Java files.

- `AccessControlList.java` - this class manages the file that stores the list of blocked callers
- `AppConfigHTTP.java` - this is the HTTP servlet that can be called from a browser to specify which callers are blocked
- `CAllBlockingSiplet.java` - this is the Siplet that performs the call blocking function

Note: You need to first use the import/export tool to export the call blocking sample application as either a standalone SAR or as part of a Enterprise ARchive (EAR) package. Then you can install it to the application server using the administrative console.

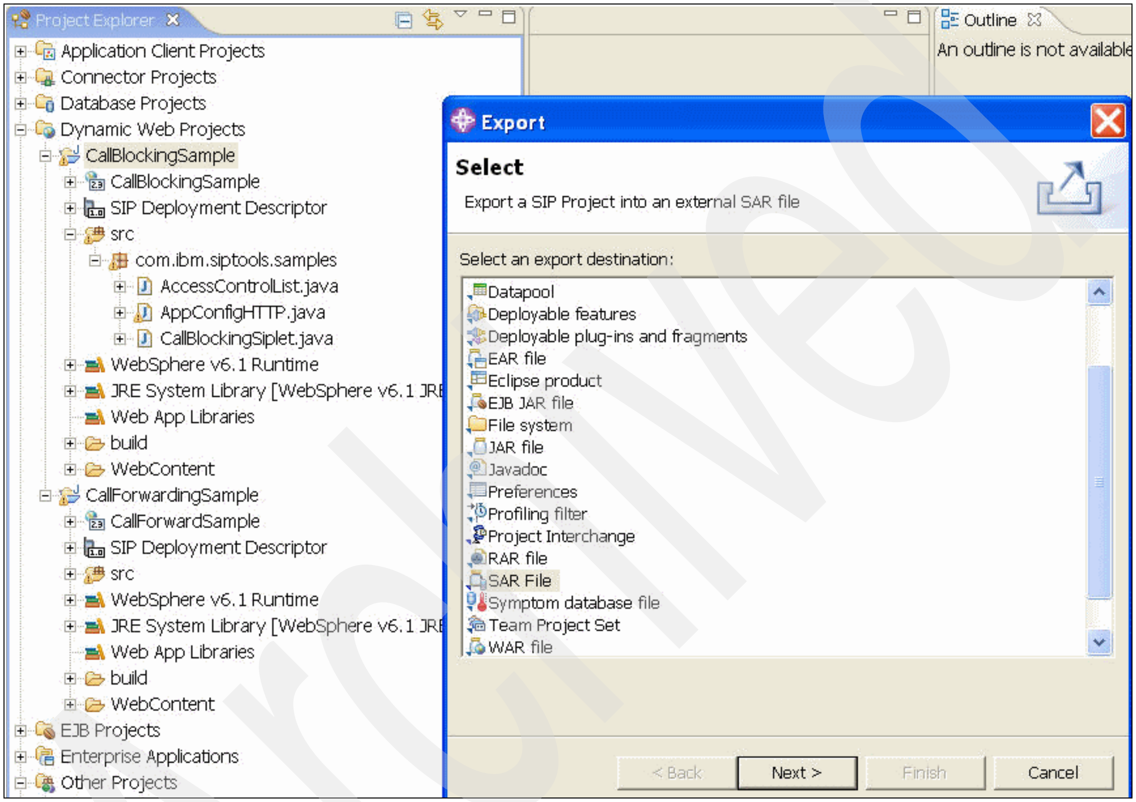


Figure 4-17 Exporting the sample

Open a Web browser and enter the URL of `AppConfigHTTP`; this is `http://localhost:9081/CallBlockingSample/AppConfigHTTP`

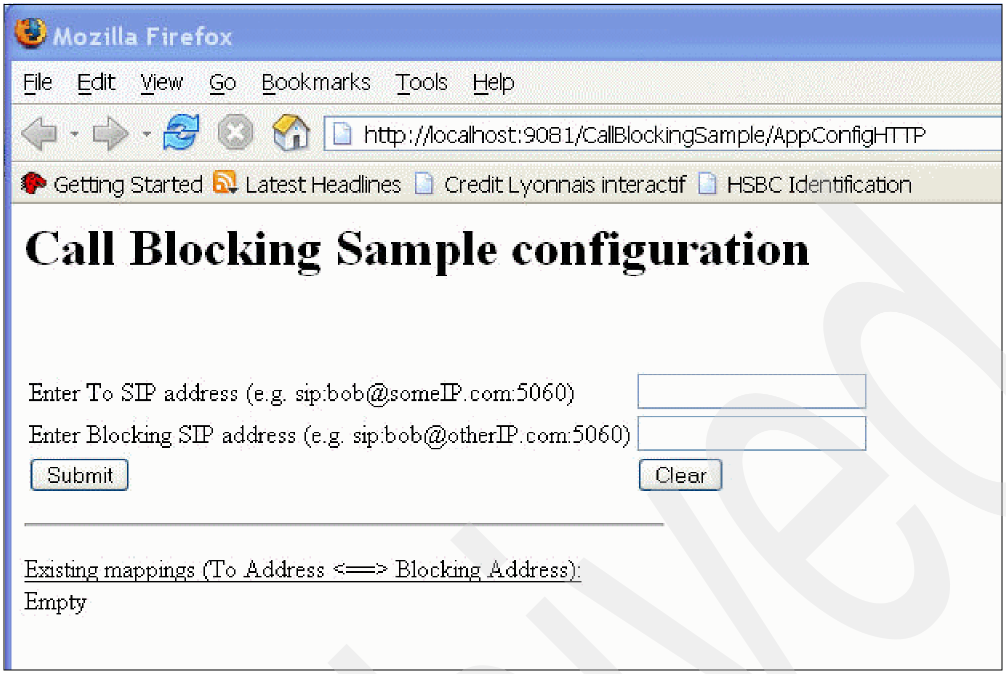


Figure 4-18 Call Forwarding Sample Web interface

> **Note:** The sample applications can be exercised by a SIP Phone or a SIP tool such as SIPp. The sample SIP applications in Part 3, "SIP applications" on page 203 show how to use SIP Phones and SIP tool such as SIPp to test SIP applications.

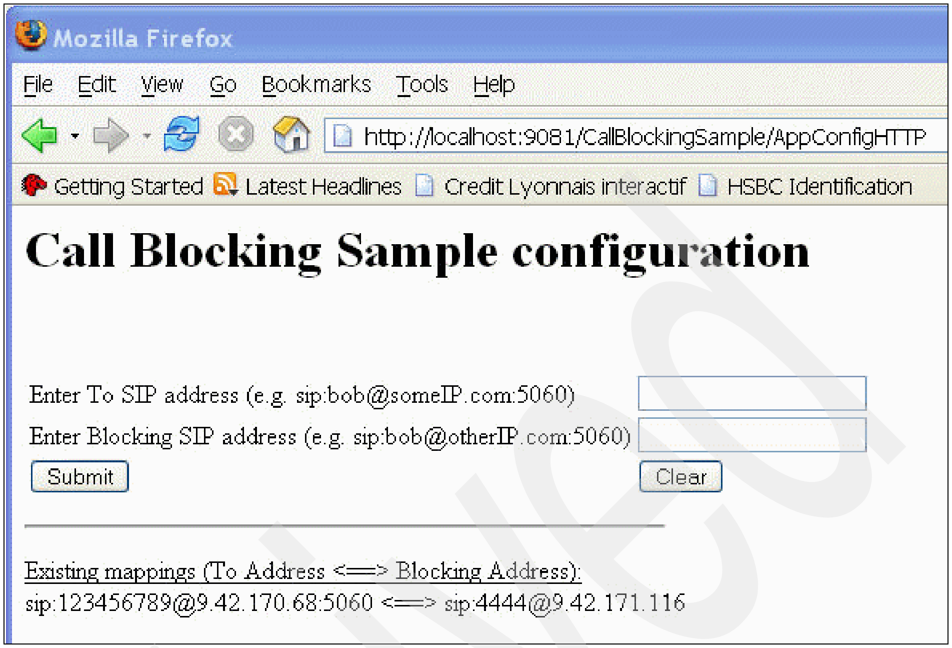


Figure 4-19 Blocking a caller

Call forwarding sample

This sample demonstrates the use of SIP Servlet 1.0 APIs. The sample performs checks to determine if the callee is in the forward list. The forward list is a simple list located in a file in the deployment package. If the callee is in the forward list, then the corresponding forward list is retrieved and the INVITE is proxied to this address.

Figure 4-20 Call Forwarding sample Web interface

This sample can be exercised by a SIP Phone or a SIP tool such as SIPp.

Third party Call Control sample

This sample demonstrates how to use converged capability by implementing a controller which sets up and manages a communications relationship between two parties.

You can exercise this sample application with two SIP phones. You initiate calls from the configuration interface `http://localhost:9081/ThirdPartyCCSample/ThirdPartyCCServer` as shown in Figure 4-21.

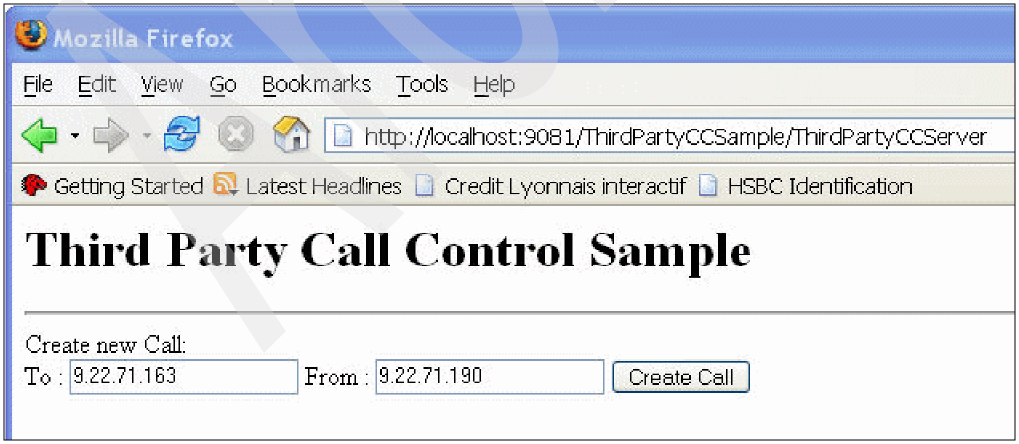


Figure 4-21 Third Party Call Control Web interface

4.2.5 Hardware and software requirements

AST runs on hardware suitable for running Microsoft Windows XP or Linux with a minimum of an Intel® Pentium® III 880 MHz processor or equivalent (1.0 GHz recommended). The following is the summary of the requirements:

- Hardware requirement

Table 4-1 AST hardware requirement

Hardware	Requirement
Processor	► Intel Pentium III 880 MHz processor ► 1.0 GHz recommended
Memory	► 512 MB RAM (minimum) ► 1 GB is recommended
Disk space	► 900 MB disk space ► 100 MB TEMP disk space is needed during installation
Display	1024x768 (minimum)

- Supported operating systems
  - Windows XP
  - Windows 2000
  - Windows Server® 2003
  - Red Hat Enterprise Linux 3.0
  - Red Hat Desktop Linux 3.0
  - SUSE Linux Enterprise Server (SLES) 9

5

IBM IMS Enablement Toolkit

This chapter provides an introductory overview of the IMS Enablement Toolkit, a set of additional resources that you add to the WebSphere Application Server Toolkit (AST) for development of IMS foundation applications that exercise Diameter, Presence and Parlay X.

This chapter contains the following:

- IMS Enablement Toolkit overview
- Developing IMS foundation applications
- Sample IMS foundation applications

5.1 IMS Enablement Toolkit overview

The IMS Enablement Toolkit consists of multiple resources (libraries and WSDLs) and samples that are loaded into your installation of the AST.

The IMS resources include the following:

- Diameter resources
  - WSDL: Creates a WSDL directory and imports Diameter WSDL files
  - Libraries: Imports libraries into the Web App Libraries directory
- Presence resources
  - Libraries: Imports presence library into the Web App Libraries directory
- Parlay X 2.1 resources
  - WSDL: Creates WSDL directory and import all Parlay X 2.1 WSDLs

Before you can import the IMS Enablement Toolkit resources you must first create a SIP project. You import the appropriate IMS resources into your project based on your application's functionality. 5.2, "Developing IMS foundation applications" on page 90 provides a walk through of the steps for including IMS resources into your SIP Project.

5.2 Developing IMS foundation applications

Start by creating a new project by selecting **File** → **New** → **Example** in AST. Select a **SIP Project** as illustrated in Figure 5-1 on page 91.

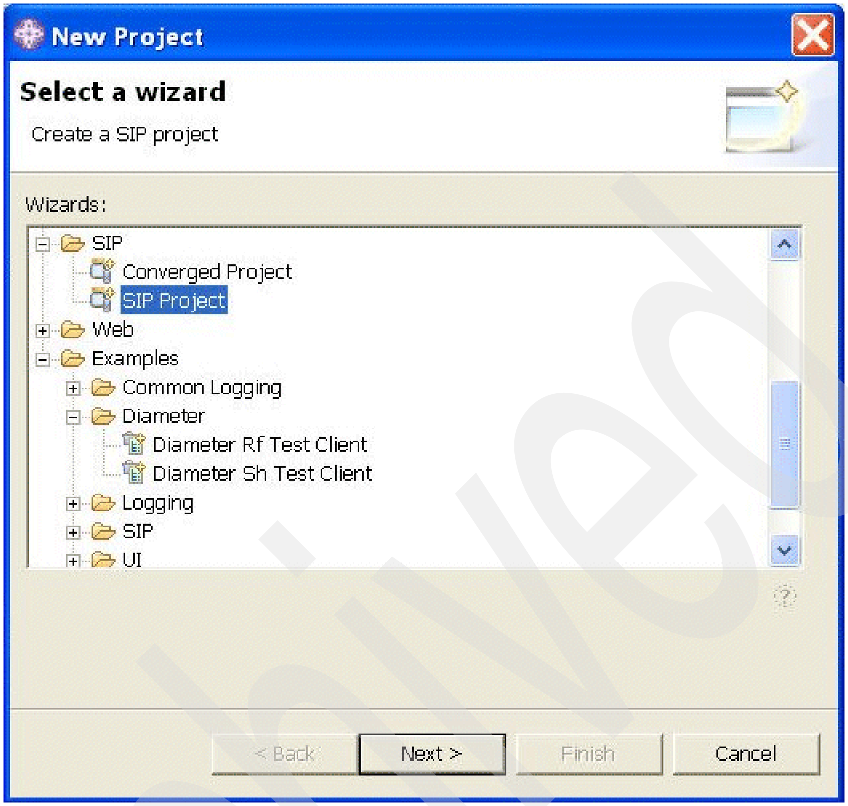


Figure 5-1 Create a SIP project

Enter the name of the project **IMS SIP Project** and click **Next**. Figure 5-2 on page 92 shows naming of a SIP project.

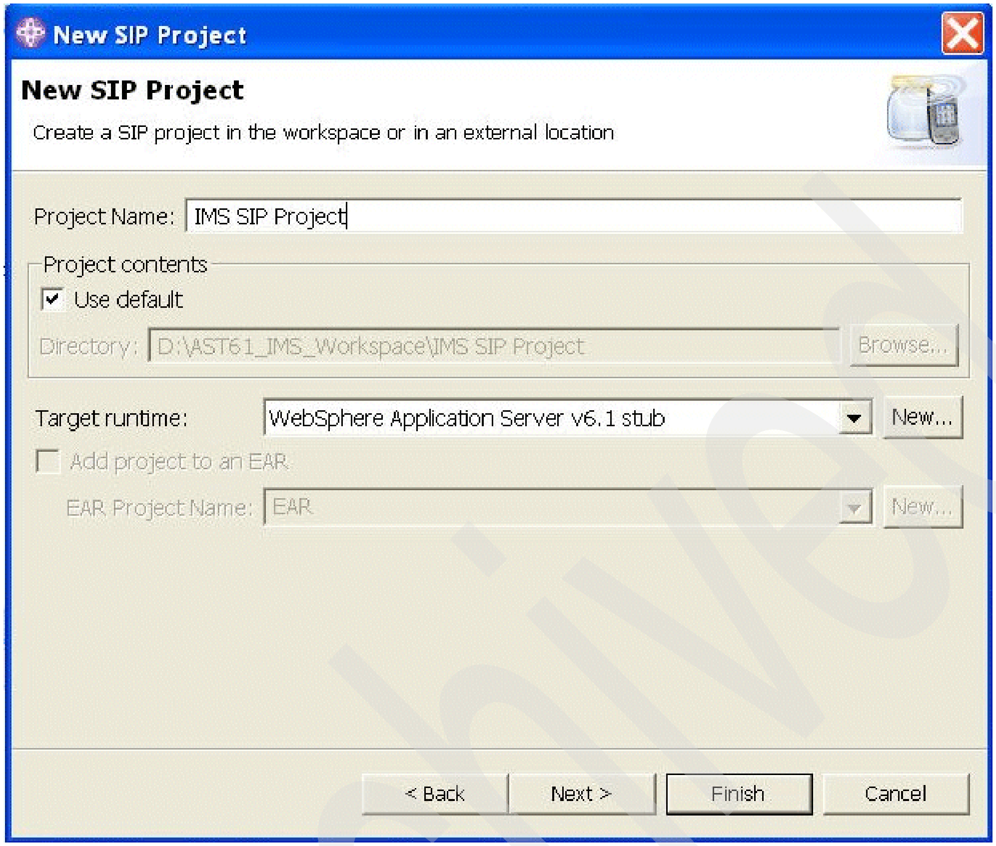


Figure 5-2 Naming the SIP project

Leave the proposed **SIP Facets** as is or change them according to your project needs. Ensure that SIPModule is selected and then click **Finish**. Figure 5-3 on page 93 shows selection of project facets.

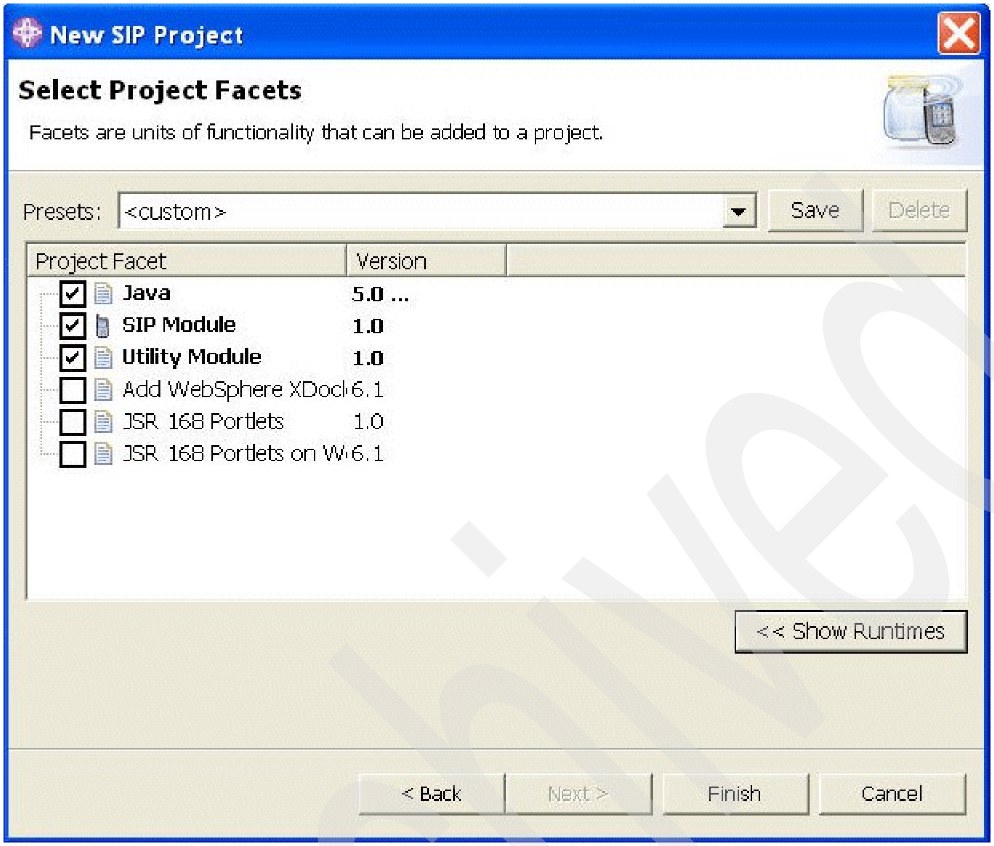


Figure 5-3 Selecting project facets

Now that the project is created, you are ready to import the IMS resources to the project. Right-click the project and click **Import.** Select **IMS Resources.** Figure 5-4 on page 94 shows selection of SIP project for IMS resource importation.

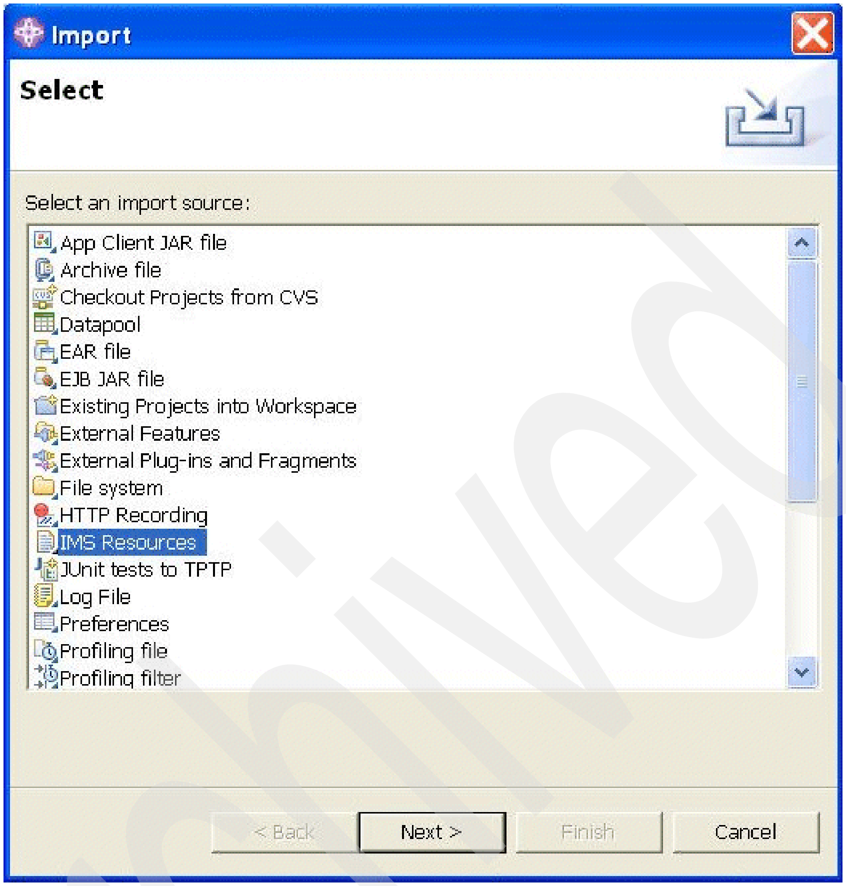


Figure 5-4 Select project for IMS resource importation

Choose the IMS Resources you want to import. Figure 5-5 on page 95 shows selection of IMS resources to import. Note that all the IMS resources in Figure 5-5 on page 95 are selected for importation. After making your selections, click **Next** to continue.

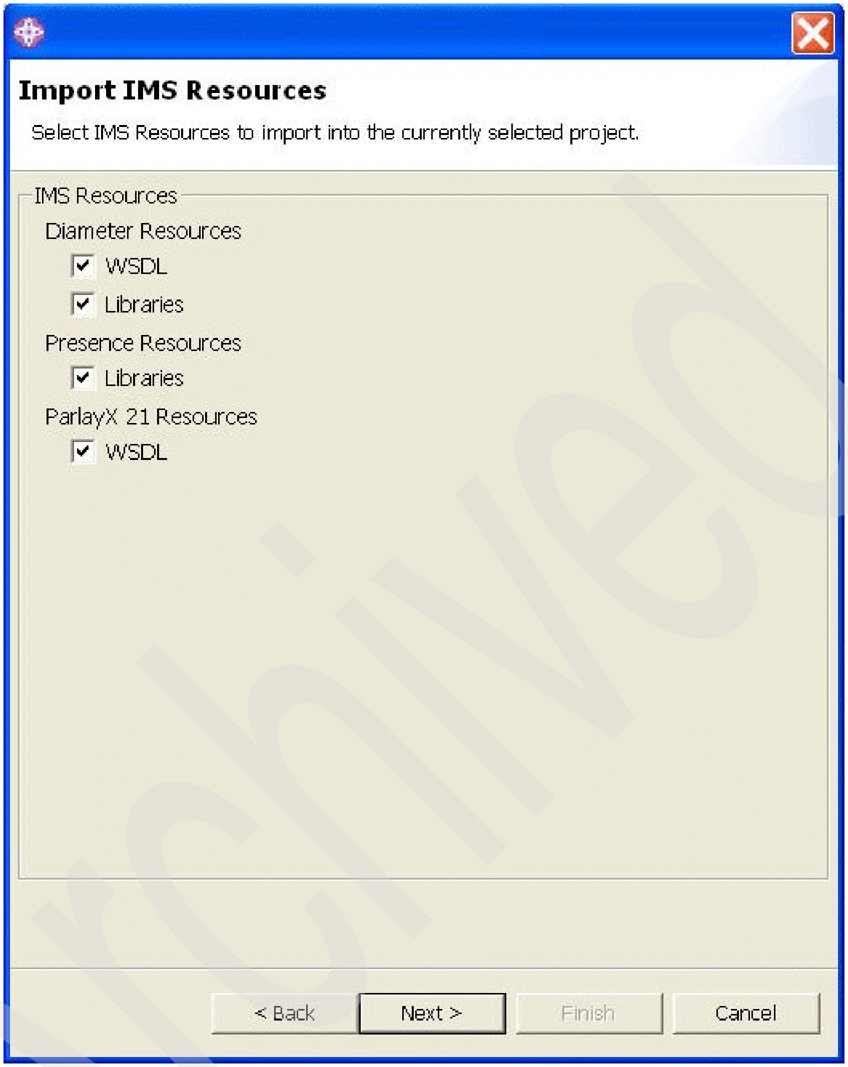


Figure 5-5 Selecting IMS resources to import

If you checked the selection for Parlay X 2.1 WSDL, you will see a list of Parlay X 2.1 WSDLs to import. Choose the ones you need for your project. Figure 5-6 on page 96 shows the selection of Parlay X 2.1 WSDLs.

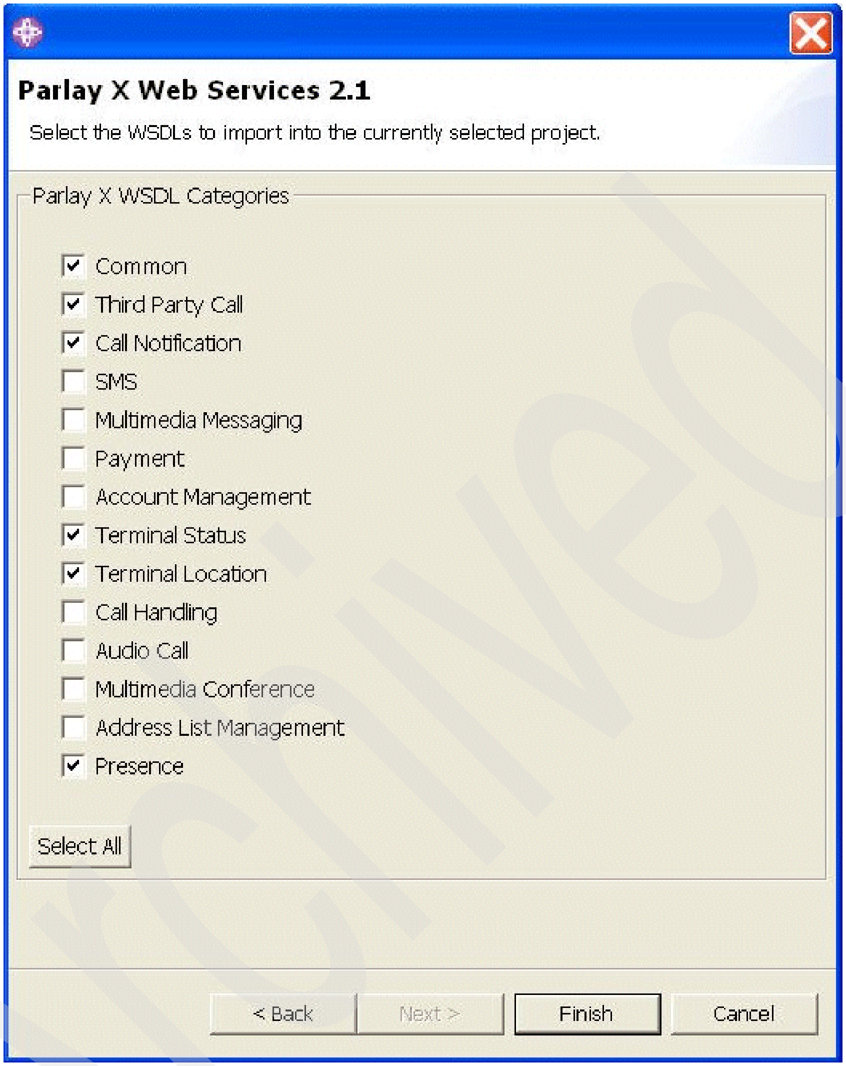


Figure 5-6 Selecting the Parlay X 2.1 WSDLs to import

Clicking **Finish** will initiate importation of the IMS resources you have selected for your project. Once the load is finished, expand the project to see the Libraries and WSDLs that is loaded. Figure 5-7 on page 97 shows the list of imported resources.

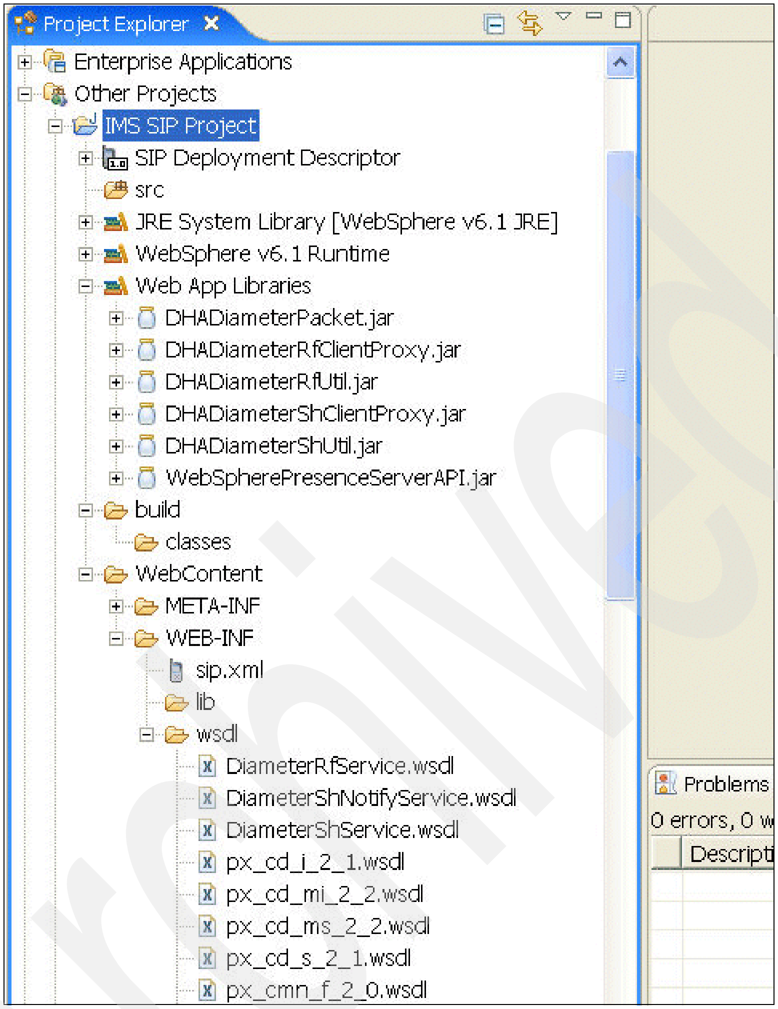


Figure 5-7 List of imported IMS Resources

With the resources loaded, you are ready to develop IMS foundation applications.

5.2.1 Diameter client application

Once the Diameter Rf and Sh libraries have been loaded as part of the installation process of the IMS Enablement Toolkit, it is possible to add the jar files in Figure 5-8 on page 98 to the Java Build Path.

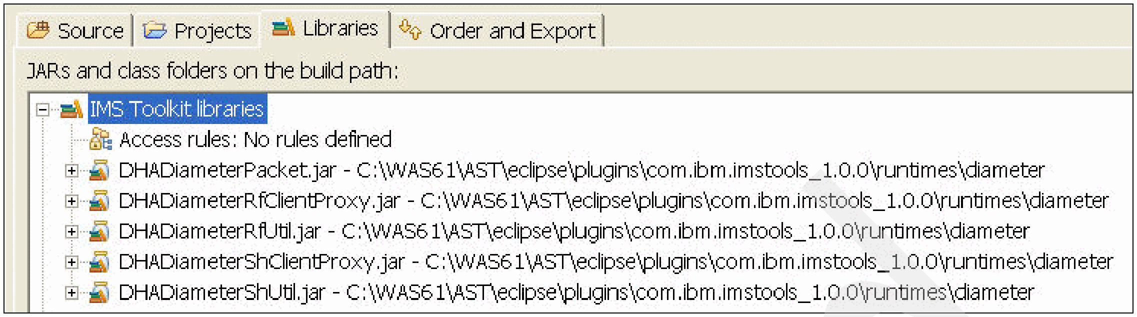


Figure 5-8 Diameter client libraries

IMS Toolkit also provides WSDLs for the following:

- Defining the Rf services used for offline charging (DiameterRfService.wsdl)
- Defining the Sh services used for subscriber profile management (DiameterShService.wsdl)
- Receiving notifications from the Sh subscriber profile Web Services when information regarding a specified subscriber has been changed (DiameterShNotifyService.wsdl)

You can explore these WSDLs in AST as shown in Figure 5-9 and Figure 5-10 on page 99. Note that for reading convenience the WSDL description has been split into the two images.

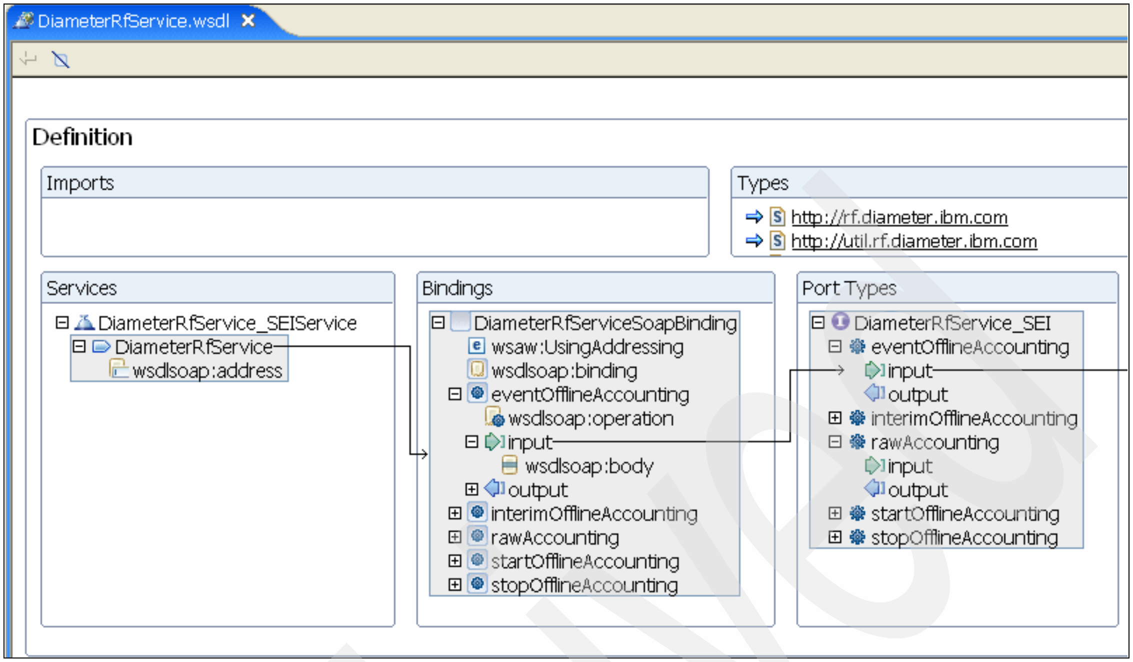


Figure 5-9 DiameterRfService.wsdl definition (part 1)

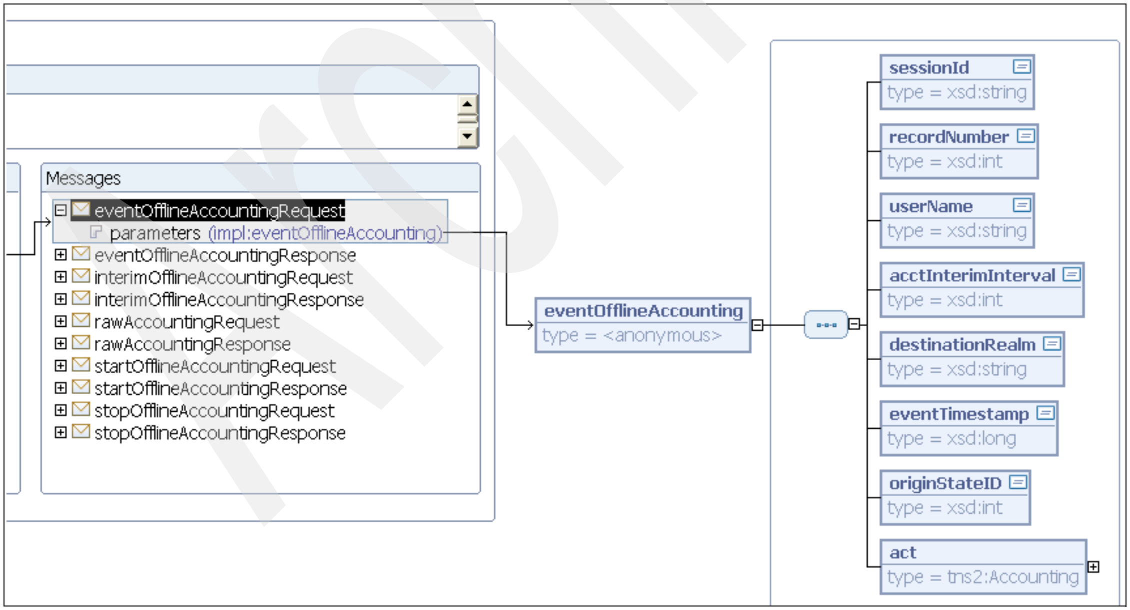


Figure 5-10 DiameterRfService.wsdl definition (part 2)

Rf accounting Web Services provide a Diameter messaging interface to enable applications to send accounting messages to accounting or billing servers. The IMS Application Server application is referred to as a client of the Web service application. The Diameter client sample (5.3.2, “Diameter client samples” on page 110) application included in IMS Enablement Toolkit is an example of how to use those Web Services to program Rf clients.

5.2.2 Presence Server components

Interfacing with the Presence Server is mainly through SIP and XCAP protocols and therefore it does not need any special tools support other than the one already provided by HTTP and SIP support. The Presence Server also has an Authorization API for allowing or disallowing subscription to a presentity. Figure 5-11 on page 100 shows the authorization classes from the IMS Enablement Toolkit presence library.

Figure 5-11 Presence Server authorization classes

Once the authorization classes are installed, Presence Server makes use of the stored permission policies and authorization API to determines if the requester, or watcher, is authorized to subscribe on a given presentity.

> **Note:** Authorization API help in AST under **IMS Toolkit, Developing, Authorization API** provides more information about how to use the API.

5.2.3 Parlay X Web Services

Support for Parlay X Web Services is provided through a set of WSDLs. The WSDLs are in categories which range from Third party call to Presence. If you chose to import Parlay X 2.1 Resources (see illustration in Figure 5-5 on page 95), you will be presented with the option to select the WSDLs that you want to import. Figure 5-6 on page 96 shows the different categories of WSDLs that are supported. You select the WSDLs that you want to import and click Finish to initiate the importation.

The sample application scenario in Part 4, “Developing IMS applications” on page 299 shows how to use the Parlay X WSDLs to invoke Web Services.

5.3 Sample IMS foundation applications

The IMS Toolkit comes with three samples applications, one IMS Service Control (ISC) SIP servlet and two Diameter clients. The samples demonstrates the usage of SIP Servlet API and ISC, and Diameter Rf and Sh test clients.

5.3.1 ISC Interface sample

The ISC SIP servlet sample demonstrates the use of SIP Servlet API and ISC to implement a Back to Back User Agent (B2BUA) service. Figure 5-12 on page 102 shows the interaction flow between the User Agents, the SIP server and the CSCF.

The ISCDemo SIP Servlet takes a call from UA1 (User Agent 1), receives INFO over this SIP session, which provides the address for UA2 (User Agent 2). ISCDemo makes outbound call to UA2 (through the S-CSCF), and then sends INFO to UA2 and UA1 separately. Then ISCDemo hangs up the call with UA1 and UA2.

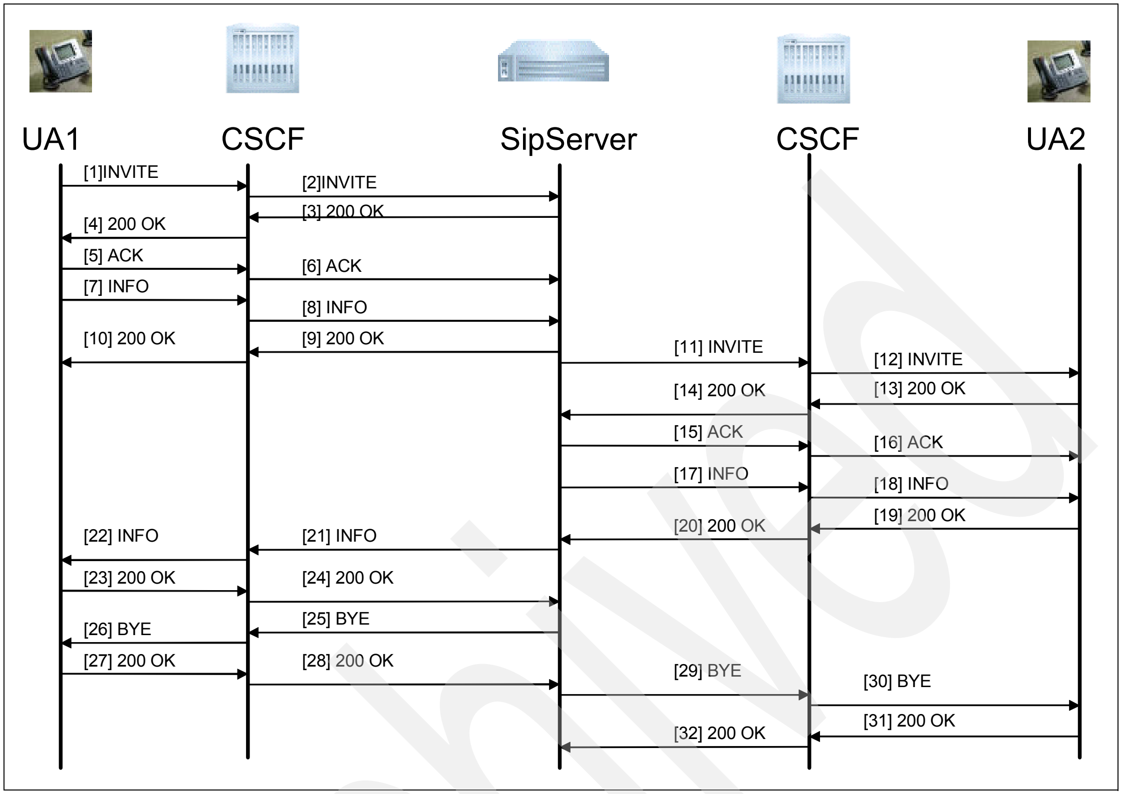


Figure 5-12 ISC Demo call flow

You load ISC SIP servlet sample by clicking **File** → **New** → **Example**.

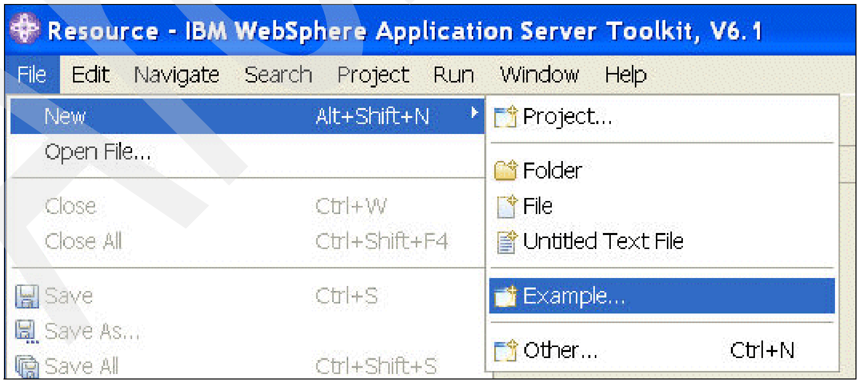


Figure 5-13 Load Example

Expand the **SIP** folder and select **ISC SIP**.

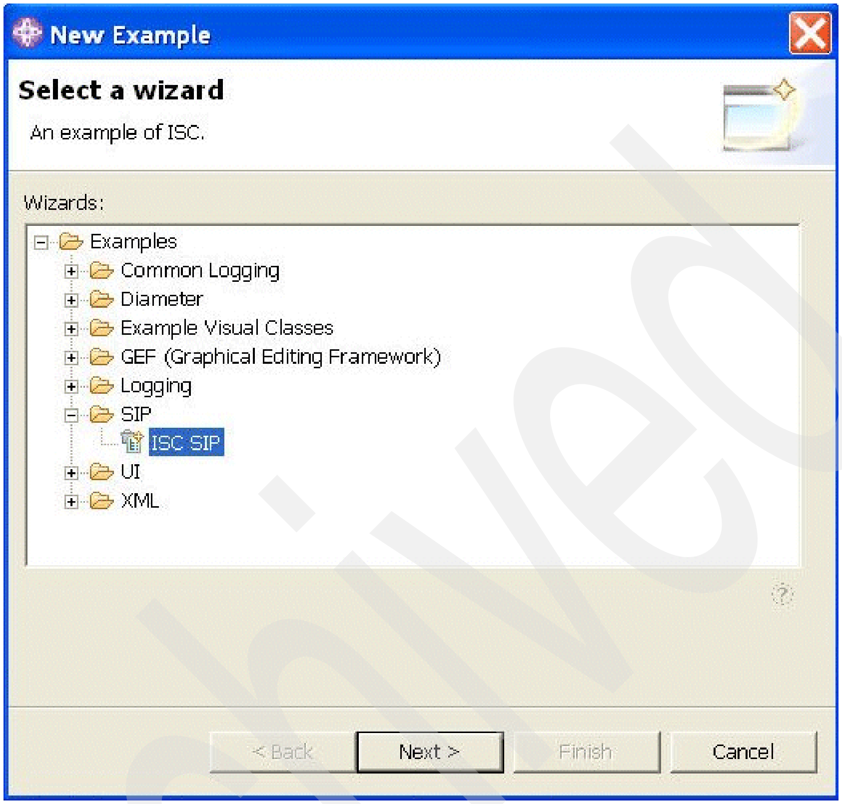


Figure 5-14 Select the ISC SIP sample

In the **ISC SIP** window, leave the default project name and click **Finish**.

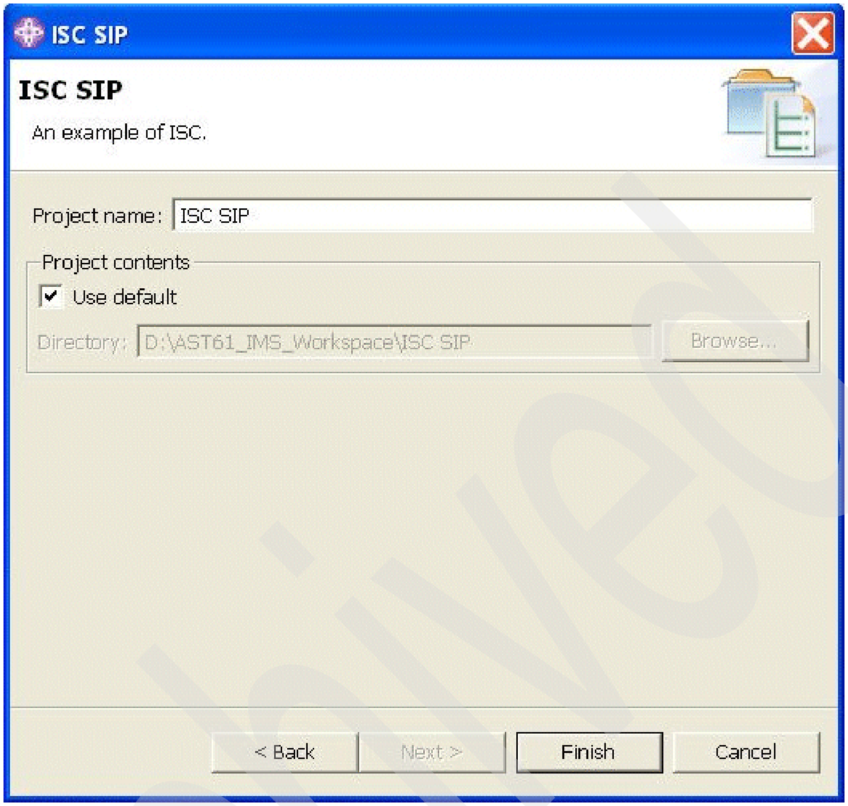


Figure 5-15 Name the ISC project

Once the loading is complete, expand the ISC SIP folder to see the Java sources, including the Java code for this example.

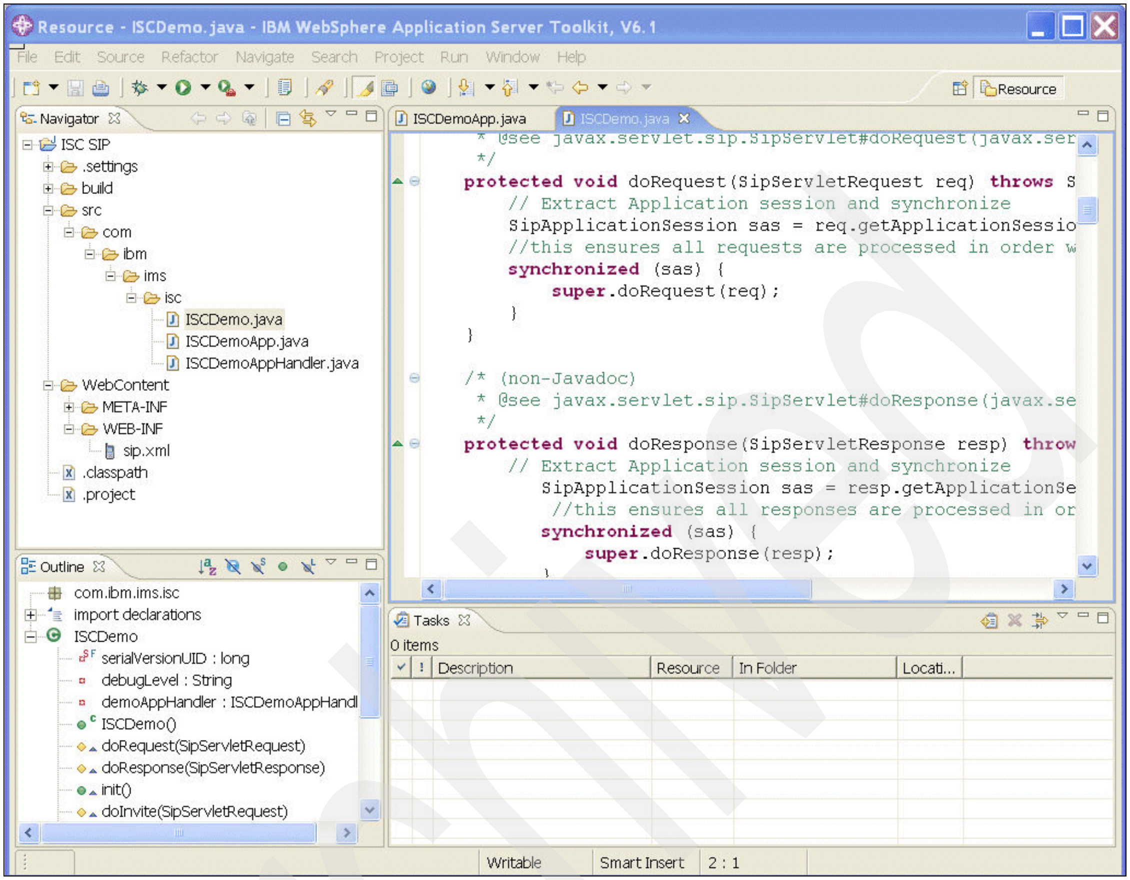


Figure 5-16 ISC SIP project in AST workspace

The ISCDemoApp states and state transition diagrams in Figure 5-17 on page 106 and Figure 5-18 on page 107 provide more information to help you understand the Java code.

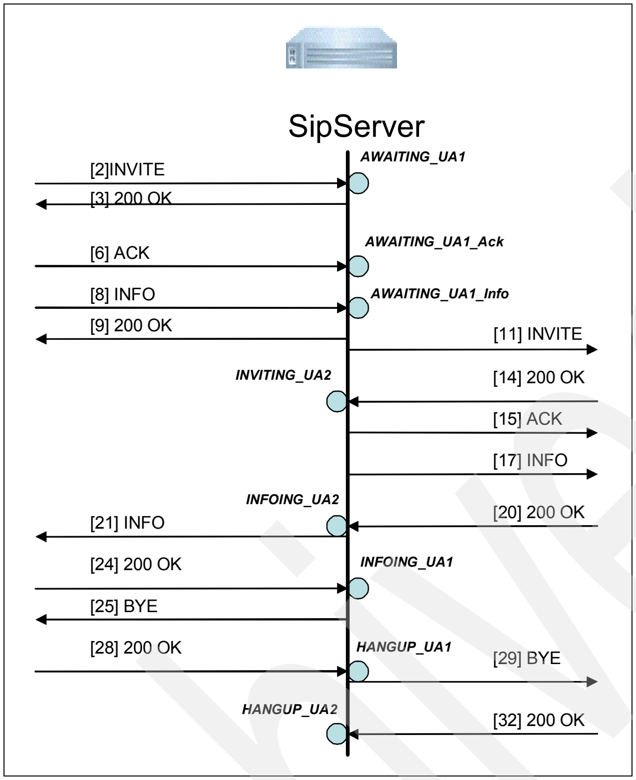


Figure 5-17 ISCDempApp states diagram

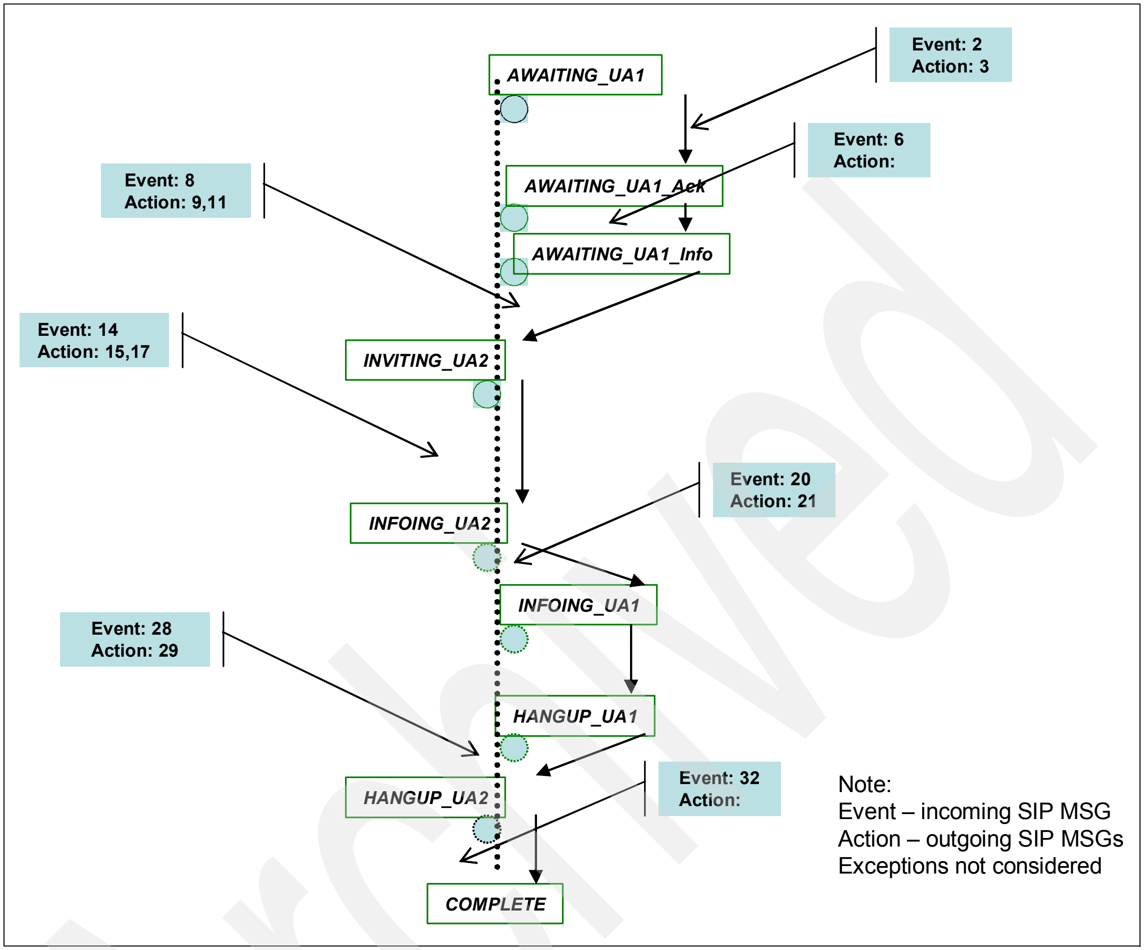


Figure 5-18 ISCDemoApp state transition diagram

ISCDemo is the SIP Servlet. It implements the `doRequest`, `doResponse`, `doInvite`, `doAck`, `doBye`, `doError`, `doInfo`, `doSuccess` methods as shown in the class outline in Figure 5-19 on page 108 and as required by the flow in Figure 5-12 on page 102.

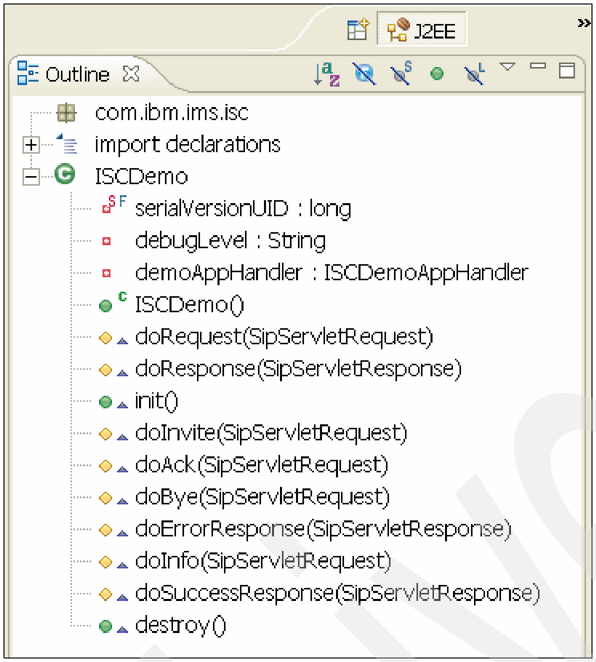


Figure 5-19 ISCDemo methods

The implementation of the doInvite() method includes constructors for ISCDemoApp and ISCDemoAppHandler, and the DemoApp state transition calls.

ISCDemoApp holds all the states and the corresponding getters and setters. When `transitState()` method is called, control is transferred to **ISCDemoAppHandler** to handle the request. Figure 5-20 on page 109 shows the methods of `ISCDemoApp`.

Figure 5-20 ISCDemoApp methods

ISCDemoAppHandler performs actions needed to transition from the current state into the specified new state. It also handles the actions for processing INVITE from UA1. Figure 5-21 on page 110 shows the methods of `ISCDemoApphandler`.

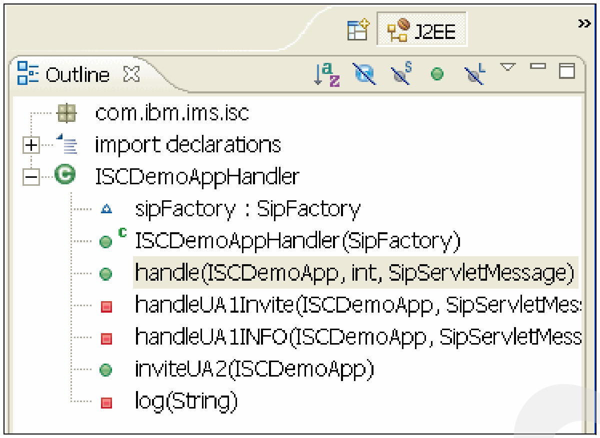


Figure 5-21 ISCDemoApphandler methods

5.3.2 Diameter client samples

Two sample Diameter clients are delivered with the IMS Enablement Toolkit;

- Diameter Rf test client which uses the offline charging WSDL DiameterRfService.wsdl.
- Diameter Sh test client which uses the subscriber profile management WSDL DiameterShService.wsdl.

Note: Both applications are similar in nature so only the walk through of the Diameter Rf test client is presented in this section.

To load the Diameter Rf test client:

1. Click **File** → **New** → **Example**.
2. Expand the Diameter folder and select **Diameter Rf Test client**.
3. Click **Next**.

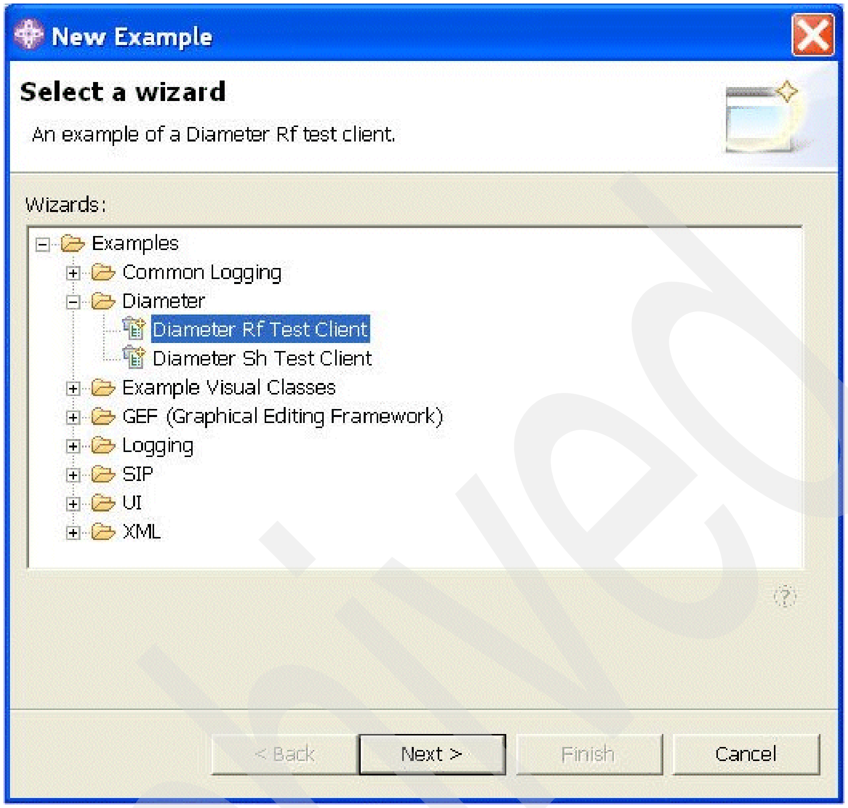


Figure 5-22 Import the Diameter Rf Test Client

Leave the default project name as is and click **Finish** as shown in Figure 5-23 on page 112.

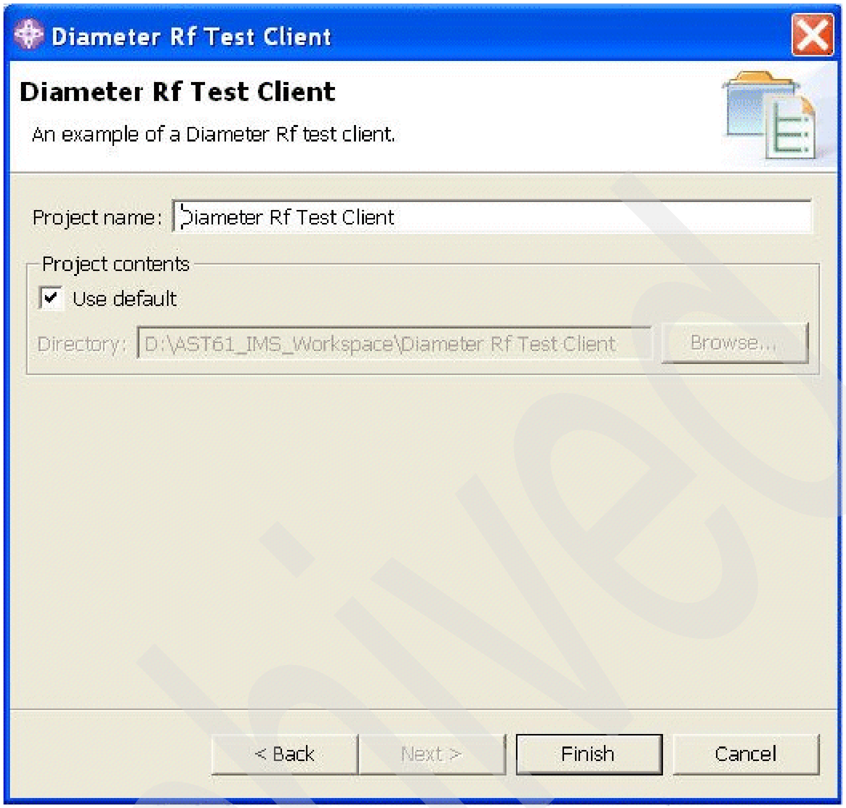


Figure 5-23 Name the Diameter client project

Once the loading is complete, expand the **Diameter Rf Test Client** in the **Navigator** pane. You may see red crosses. If you double-click **DHADiameterRfTestClient.java**, the import of `javax.xml.rpc.Stub` has a red cross in front of the line as illustrated in Figure 5-24 on page 113. This indicates that you have libraries that have yet to be included to your project.

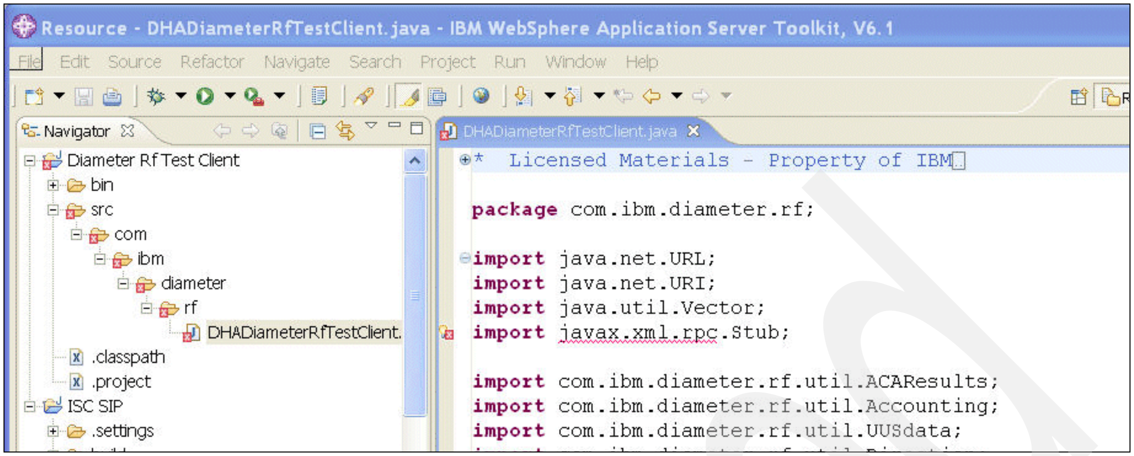


Figure 5-24 Missing imports in Diameter RfTest Client

Now let's check the Java Build Path. Right-click the project name and select **Properties**. Select the **Java Build path** and **Libraries** tab. You can expand the IMS Toolkit libraries and verify that the Diameter client libraries have already been added to the path. Figure 5-25 on page 114 shows an illustration of the JARS and class folders on the Java build path.

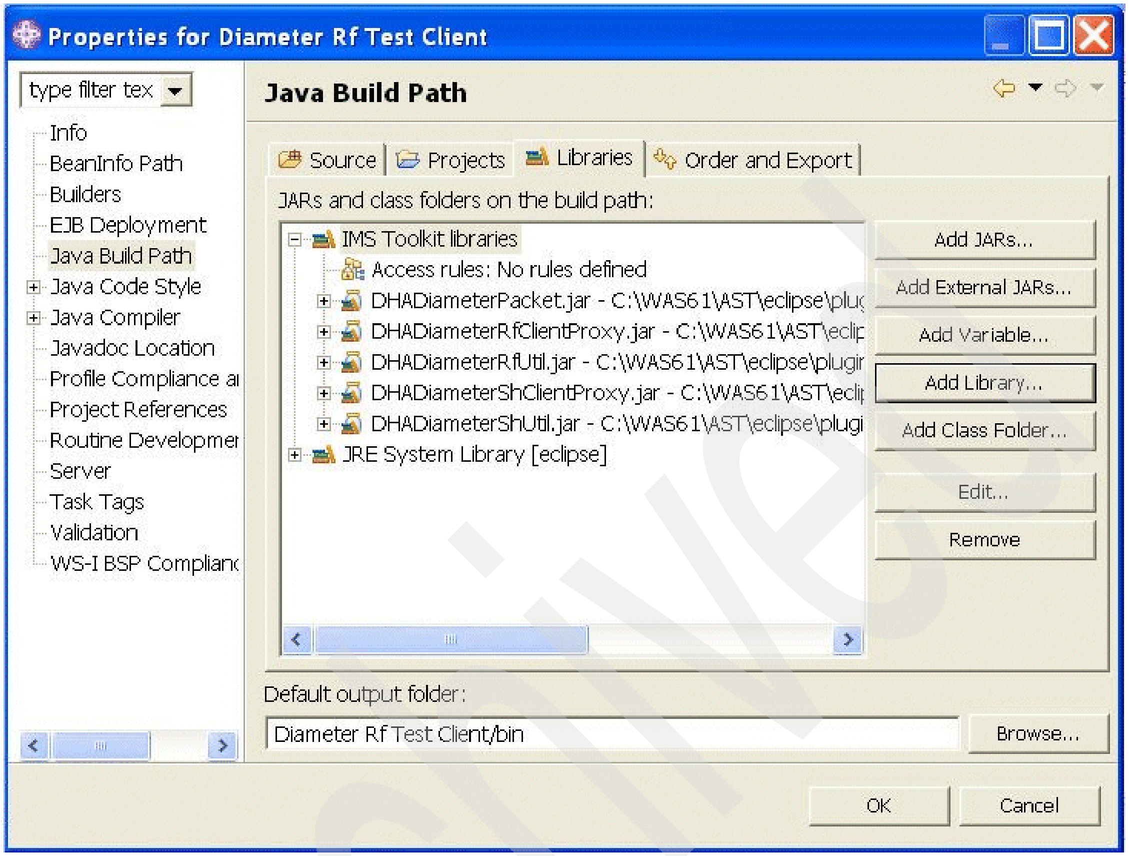


Figure 5-25 Modify the Java Build path

If it is the case that the IMS Toolkit libraries are not already on the build path, you can add it by clicking **Add Library** button and selecting **IMS Toolkit Libraries.** Click **Next** to continue. See an illustration in Figure 5-26 on page 115.

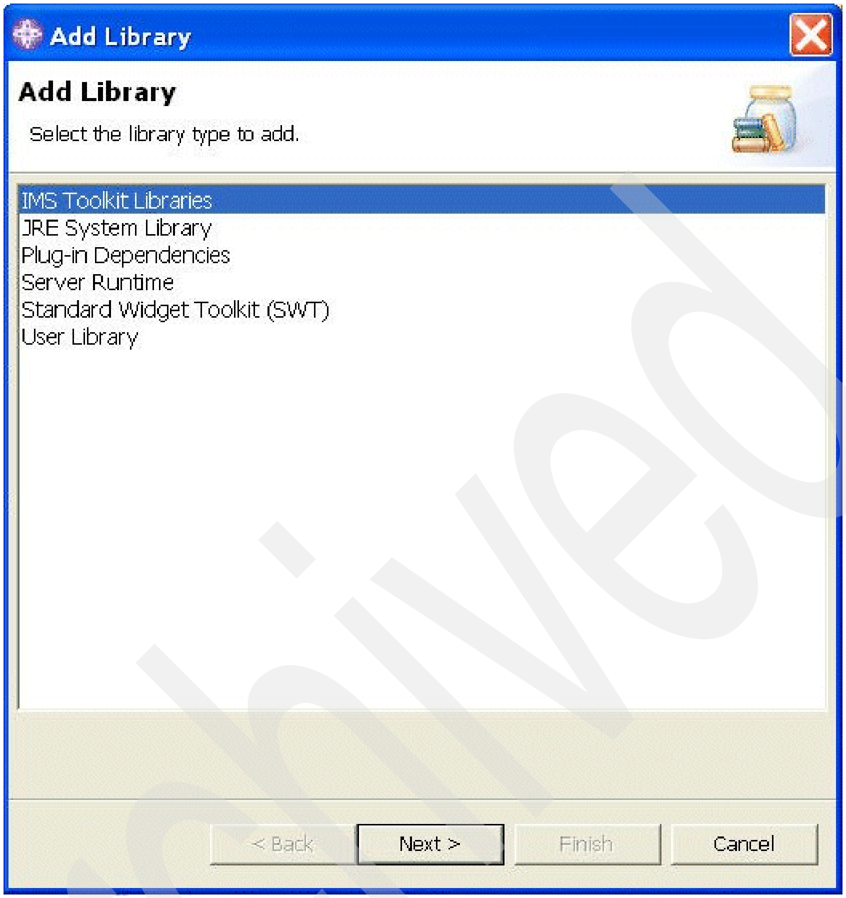


Figure 5-26 Add a Library to the Java Build Path

You can then select Diameter and/or Presence libraries and click **Finish** as in Figure 5-27 on page 116.

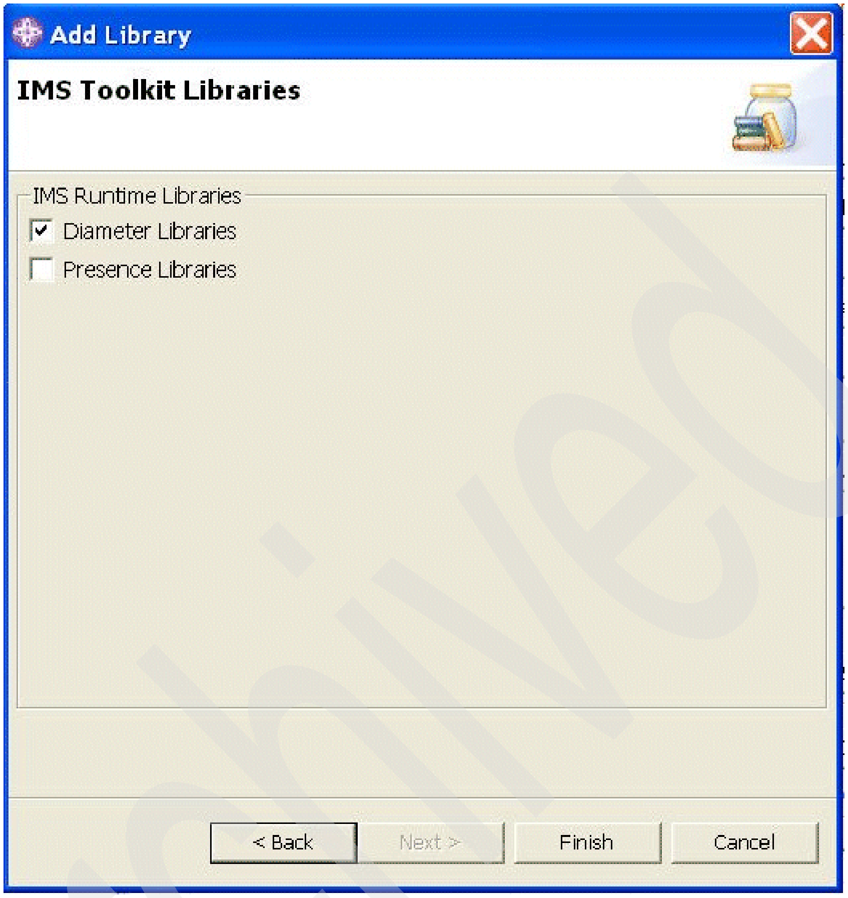


Figure 5-27 Add IMS Toolkit Libraries

Here, we chose the **WebSphere Application Server v6.1 stub** by selecting it and clicking **Finish.** See the illustration in Figure 5-28 on page 117.

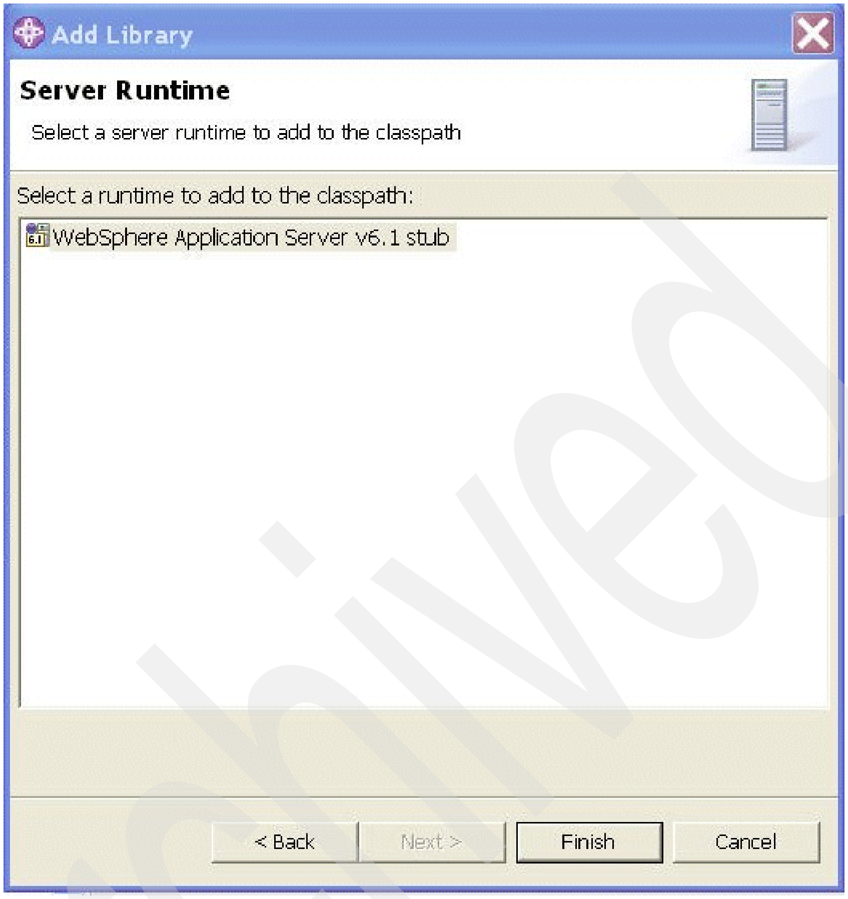


Figure 5-28 Add WebSphere Server runtime libraries

The WebSphere APIs JAR file will be added to your project (see illustration in Figure 5-29 on page 118) and you should no longer have the red crosses as in Figure 5-24 on page 113.

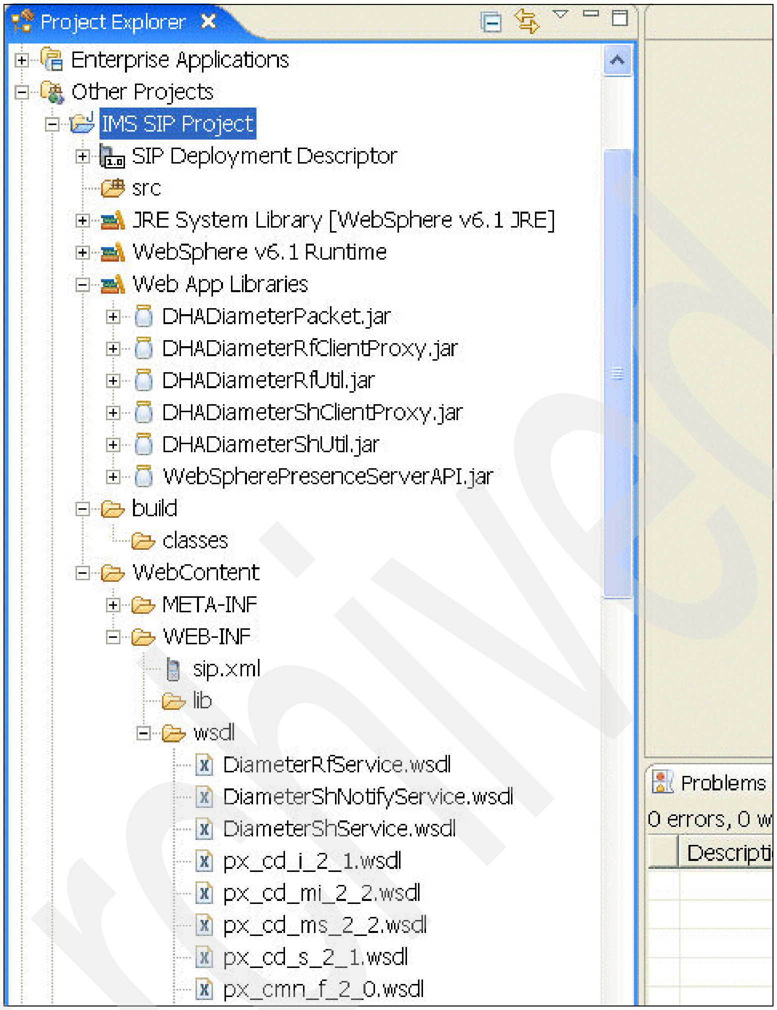


Figure 5-29 Summary of Java Build Path

> **Note:** The process of loading the **Diameter Sh test client** is identical, so you have to perform similar operations for **Diameter Sh test client.**

6

IBM WebSphere Integration Developer

WebSphere Integration Developer (WID) provides the composite applications development platform for IBM Telecom Web Services Toolkit. In this chapter we introduce WID, the key concepts and components for creating and assembling services.

This chapter contains the following:

- Overview
- Working with IBM WebSphere Integration Developer
- IMS service components
- Technical information

6.1 Overview

WebSphere Integration Developer (WID) is an Eclipse-based visual application development environment for creating integrated applications based on service-oriented architecture. WID enables both the creation and assembling of component services. These services include business rules, human tasks, business state machines, and mediation flows.

WID is designed to be used by integration specialists to assemble component services into new services and applications. WID visual editors provide a layer of abstraction between the interfaces of the components and the actual implementations. By dragging and dropping the components into the visual editors and wiring their interfaces together, the integration specialist is able to create services and applications without detailed knowledge of the underlying implementation of each component.

WID supports both top-down design approach where the implementation for one or more components does not already exist and is added later, and bottom-up approach where the implementation of the components already exist. The integrated services comply to industry-wide standards, for example, business processes which are composed of components comply to the industry-standard Business Process Execution Language (BPEL).

WID supports modeling, building, testing and debugging of solutions that deploy to WebSphere Process Server and WebSphere Enterprise Service Bus.

6.2 Working with IBM WebSphere Integration Developer

WebSphere Integration Developer is based on a number of key concepts. In this redbook, we focus on those concepts which are relevant for integration in the context of IMS applications.

> **Note:** We recommend reading the IBM redbook *Getting started with WebSphere Integration Developer and WebSphere Process Server*, SG24-7130, for a complete and detailed introduction to the WebSphere Integration Developer and WebSphere Process Server business integration concepts and tools.

6.2.1 Key concepts

The WebSphere Process Server SOA model defines three abstractions that are key to any integration project. Figure 6-1 on page 121 provides an illustration of these concepts. The concepts in conjunction with the intuitive graphical environment of the WebSphere Integration Developer offer powerful tools for the process designer and integration developer.

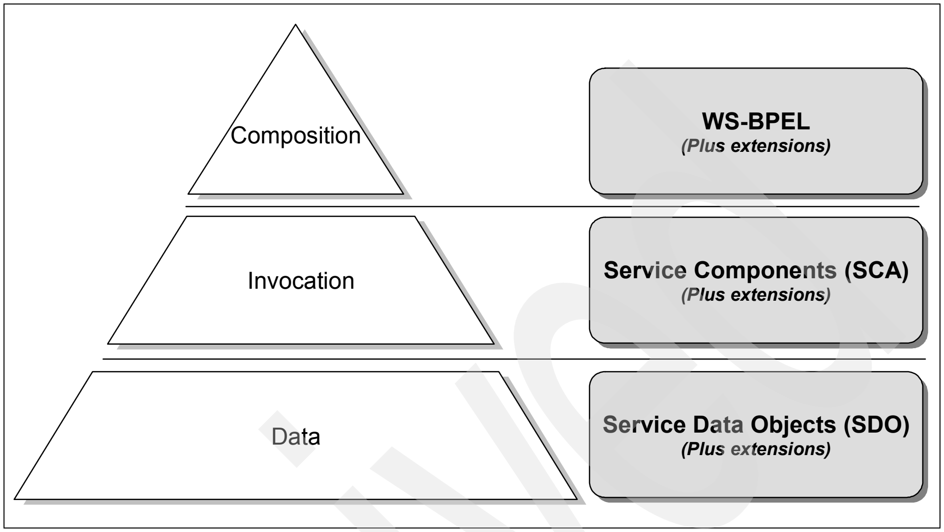


Figure 6-1 WebSphere Process Server key abstractions

- Data objects

  Service Data Object (SDO) is used to define the data level concept. SDOs can describe complex data structures and provide the universal means for representing and accessing data, as well as for transferring data between components. SDOs coupled with a number of extensions are used to implement Business Objects. WebSphere Process Server define SDO as an abstract concept, and WebSphere Integration Developer represent SDOs as business objects.

- Invocation and integration

  Service Component Architecture (SCA) provide the invocation level concept for integrating artifacts as service components with well-defined interfaces. SCA also introduces the concept of modules, which groups together service components and provides further specification and encapsulation of services.

  WebSphere Process Server supports the Service Component Architecture. It provides integration artifacts such as processes, business rules and human tasks. WebSphere Process Server support of SCA makes it possible for example to replace a human task for approval with a business rule by simply replacing the service components in the assembly diagram without changing either the business process or the invocation of the business process.

- Composition

  The composition level concept of orchestrated composite services is realized in WebSphere Process Server by Business Process Choreographer which provides support for business processes and human tasks. It supports business process modeling based on Web Services Business Process Execution Language (WS-BPEL or BPEL). BPEL is a model and a grammar for describing the behavior of business processes based on interactions such as human-to-human, human-to-machine, and machine-to-human.

6.2.2 Modules

Modules consist of service components, imports and exports which reside in the same project and root folder. Modules also contains the wiring that links the components and the bindings needed for the imports and exports.

There are two types of modules:

- Module

  Also referred to as a business integration module. It contains choice of many component types often used to support business processes. Modules deploy to the WebSphere Process Server.

- Mediation module

  Contains one component, a mediation flow component, plus zero or more Java components that augment the mediation flow component. Mediation modules can be deployed to either the WebSphere Process Server or the WebSphere Enterprise Service Bus server.

Module and mediation module artifacts include:

- Module definitions

  Defines the module.

- Service components

  Service components define the services in the module. The name for a service component must be unique in a module, however a service component can have an arbitrary display name, which is typically a name more useful to a user.

- Imports

  Imports define the calls to services external to this module.

- Exports

  Exports are used to expose components to callers that are external to this module.

- References

  Refer to components in other modules.

- Stand-alone references

  Stand-alone references are applications that are not defined as Service Component Architecture components (for example, JavaServer™ Pages™), which enable these applications to interact with Service Component Architecture components. There can be only one stand-alone reference artifact per module.

- Other artifacts

  These artifacts include WSDL files, Java classes, XSD files and BPEL processes.

Figure 6-2 shows the illustration of a simple service module. The service component is implemented as BPEL process. It exposes an interface `Export`. The service component imports the services of a component called `Import` and the service of another service component, that implements a human task.

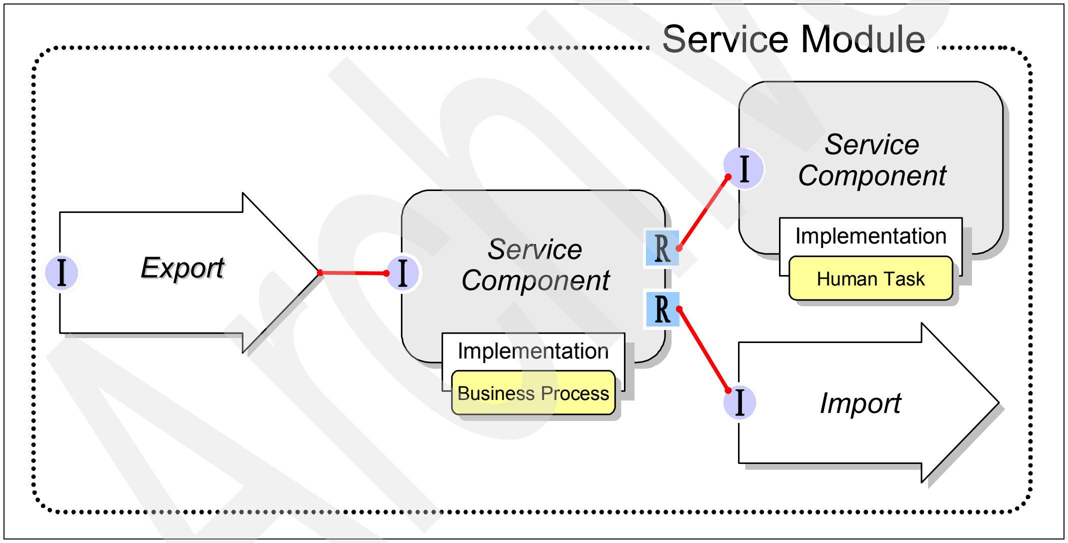


Figure 6-2 Simple service module

6.3 Components

Business integration modules contain interconnected SCA components. The WebSphere Integration Developer support seven different types of components, which are described in Table 6-1 on page 124. All component types adhere to the same component architecture but are realized using different technologies.

Table 6-1 SCA components

Component	Description	Applicable for IMS?
Process	Implements BPEL process	Yes - this is the most common implementation
State machine	Implements a state machine	Typically the SIP servlet implements the SIP protocol state machine. Use with care. It is difficult to coordinate two interacting state machines.
Human task	Implements human activities	Yes - if human interaction is part of the application
Java	A Java component	Yes - in cases where J2EE components are part of the service.
Rule group	Implements a set of business rules	Yes.
Selector	Rules-based, dynamic selection of import Web Services at runtime	Yes.
Interface map	Mapping between interfaces	Yes.

The structure of an SCA component is illustrated in Figure 6-3 on page 125.

A component consists of the following:

- Implementation

  Implements the component functionality using a choice of various languages. Implementations are hidden from WID visual editors.

- Interface

  Components contain one or more interfaces which define the inputs, outputs and faults. Interfaces support asynchronous and synchronous interaction styles and may be defined in one of two languages: a Web Services Description Language (WSDL) or Java.

  Note: You cannot mix WSDL and Java definitions for a given interface.

- References

  References identify the interface of other services or components that this component requires or consumes. A component can contain zero or more references.

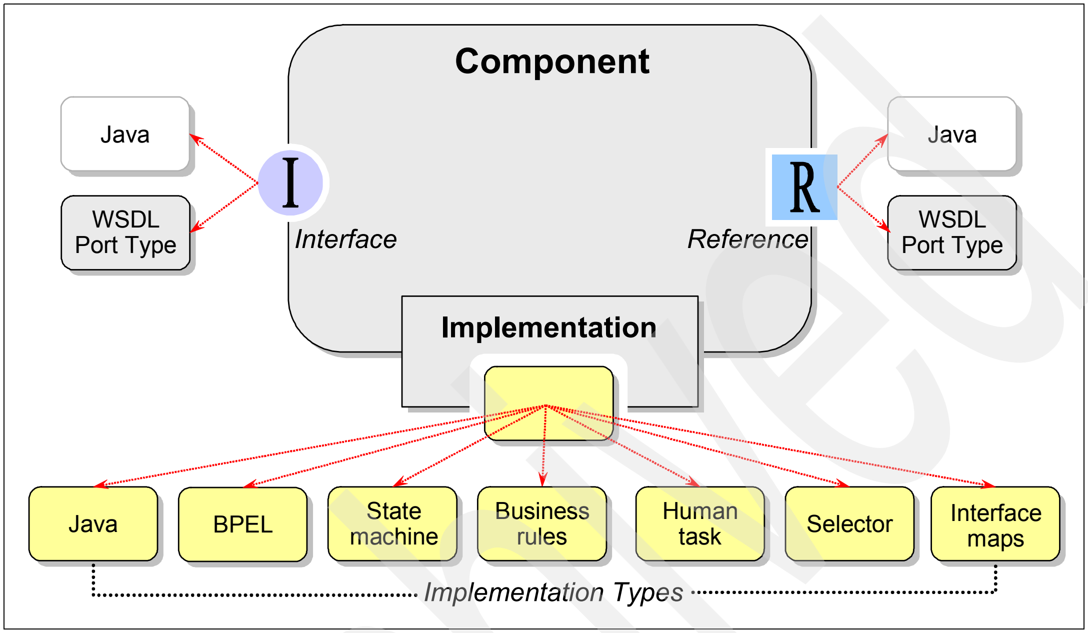


Figure 6-3 Structure of SCA component

There a three pseudo-components. They differ from the other components in that they don't have an implementation that is executable by the WebSphere Process Server. The pseudo-component is basically an SCA-wrapper around these non-SCA implementations. This way they are available to the assembly editor, where they can be wired to the other components. The pseudo-components are described in Table 6-2.

Table 6-2 SCA pseudo-components

Component	Description	Applicable for IMS?
Import	References external services	Yes - SIP servlet is a representation of import component
Export	Exposes the interface of the module to the outside world	Yes - all enablers and foundation services that require choreography are representations of export component

Component	Description	Applicable for IMS?
Stand-alone reference	Non-Web service client of the module	Most likely not - there should be a single point of entry to the IMS application and this is the SIP servlet. The converged container enable access from J2EE applications through the SIP servlet.

Both, import as well as export components require a binding, which specify the means of transferring data from the modules. An import binding describes the specific way an external service is bound to an import component. An export binding describes the specifics of how a module's services are made available to clients. Typically in the IMS world, the bindings would point to the Web Services endpoint. You may also generate bindings to JMS queues or other SCA components.

> **Note:** You can find more information about Service Component Architecture in the WebSphere Integration Developer Information Center at:
>
> `http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wbit.help.prodovr.doc/topics/cservcomps.html`

6.3.1 Business Integration perspective and views

The WebSphere Integration Developer adds a new perspective to the Eclipse framework, the Business Integration perspective. Figure 6-4 on page 127 shows the main view of the Business Integration perspective.

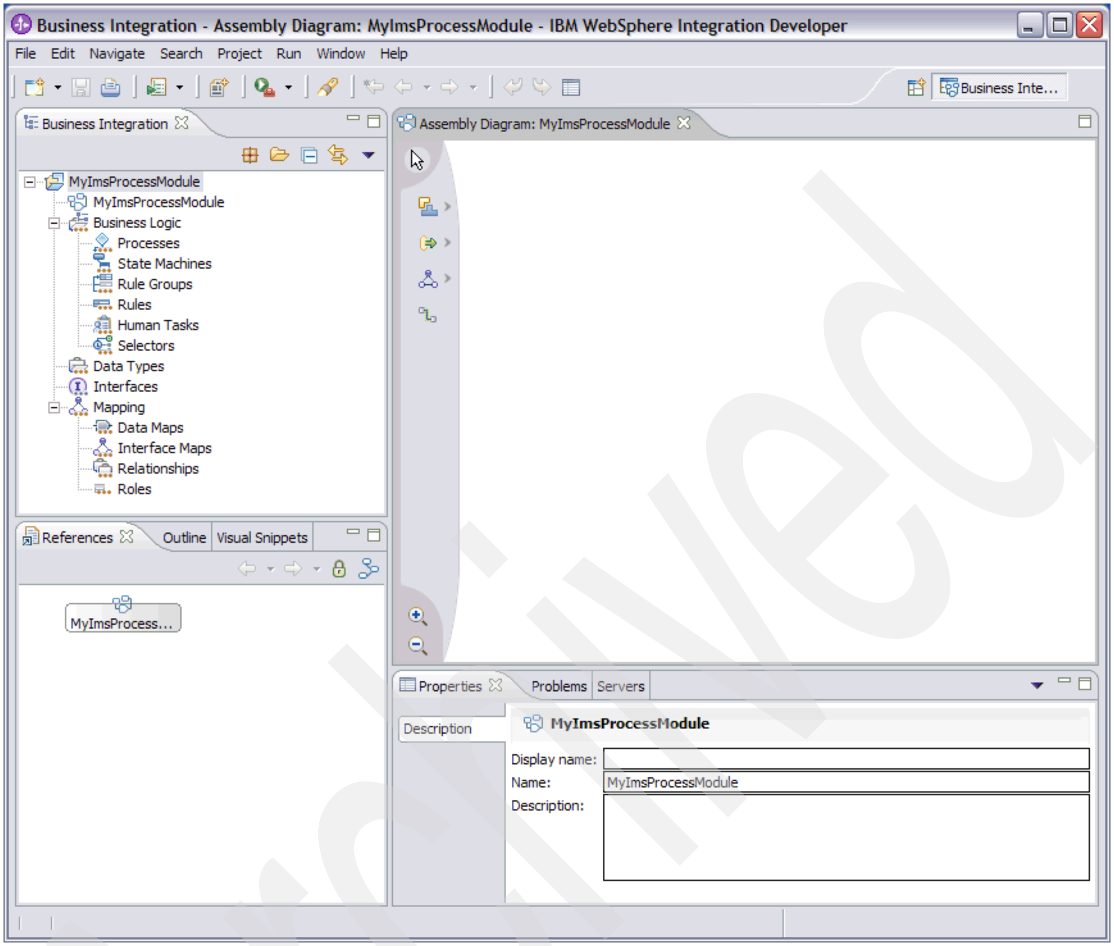


Figure 6-4 Business Integration perspective

The main view of this perspective is the Business Integration view, which provides the outline for all integration artifacts. Let us take a closer look at the structure as illustrated in Figure 6-5 on page 128.

- MyImsProcessModule

  This entry represents the module. Double-clicking it will open the assembly diagram. This folder contains references to import and export components.

- Business Logic

  These sub-folders contain the implementations of the components. For example the Processes folder contains BPEL processes. Each sub-folder contains specific diagrams and editors.

- Data Types

  This folder contains all business objects that you define in your project, as well as those that are imported from external interfaces (for example, WSDL files and the types defined in `.xsd` schema files). This folder also contains the business object diagrams and editor.

- Interfaces

  This folder contains all interfaces, including the imported and exported interfaces.

- Mapping

  This folder and the sub-folders contains supporting components such as Data Maps for mapping business objects to one another, Interface Maps for mapping interfaces, Relationships for defining relationships between business objects as well as Roles.

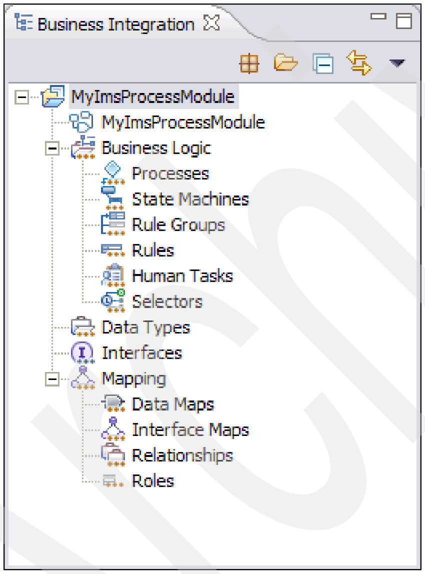


Figure 6-5 Business Integration view

6.3.2 Adding custom logic to BPEL processes

Various activities in a BPEL flow require the specification of custom logic. For example:

- The while loop activity require that you specify the exit condition.
- The choice activity require that you specify the conditions to follow a branch.
- The snippet activity enable you to specify custom logic.

You have two options for specifying the logic

- By writing Java code.
- By using a visual editor to specify the logic graphically.

> **Attention:** You can use either Java code or graphical representation to specify the logic. You cannot mix and match the two methods. Switching from one method to the other will result in you loosing the logic you have created so far.

Figure 6-6 shows an illustration of what the visual editor looks like. To create logic using the visual editor, you drag and drop the snippets onto the canvas and wire them together by grabbing the yellow link icon and dragging it to the desired target snippet.

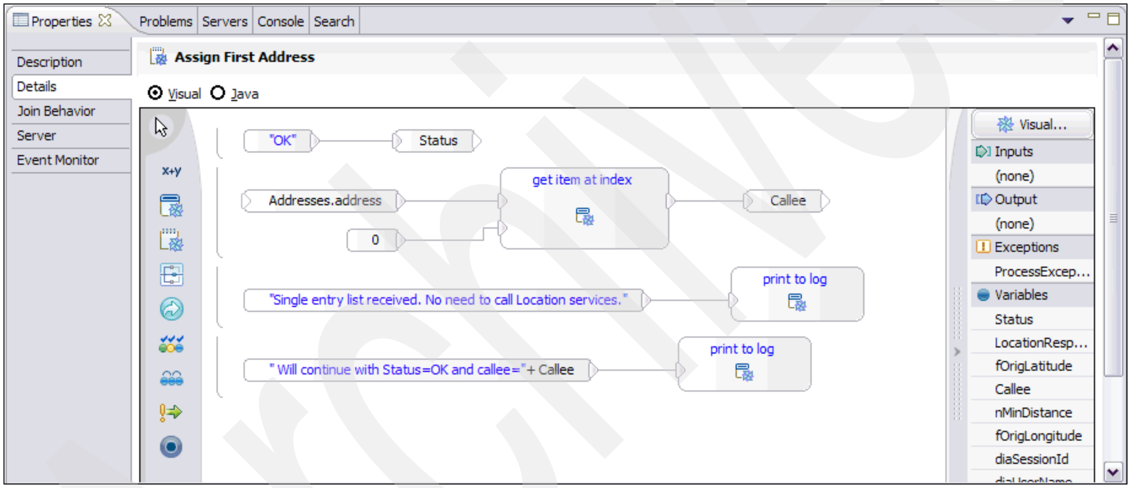


Figure 6-6 Visual snippet editor

Let us introduce the key elements of the visual editor:

- Editor selector buttons

  The radio buttons at the top of the canvas enable you switch between the Java and the visual editors.

- Tools palette

  The tools palette to the left of the canvas provides you quick access to the following:

  – Expression® editor
  – Set of standard snippets you can choose from

- Java classes and interfaces
- Choice construct
- While construct
- For each construct
- Repeat construct
- Throw snippet to throw exceptions
- Return snippet

- Visual panel

  The panel on the right side of the canvas shows Inputs, Outputs, Exceptions and Variables that are available to the snippet. They are inherited from the process scope.

The custom Java logic in Figure 6-7 performs the same function as the visual snippet in Figure 6-6 on page 129.

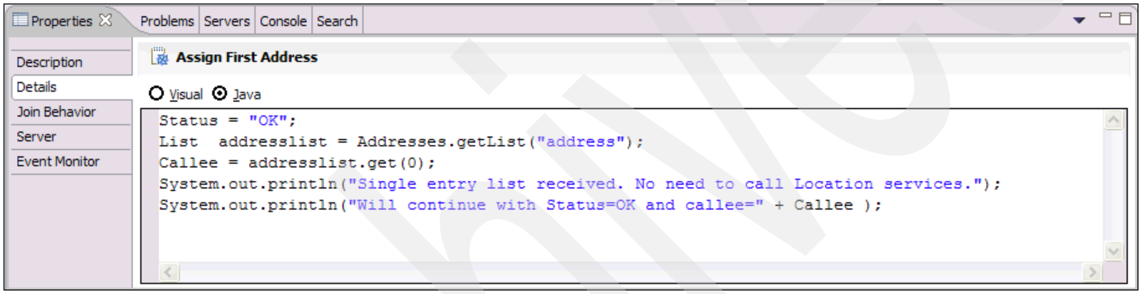


Figure 6-7 Java snippet

> **Important:** To import the required Java libraries, click anywhere on the business process editor canvas. In the Properties view of the business process select the tab **Java Imports**. In there define the import statements like: `import java.util.List;`

6.4 IMS service components

IMS applications developed using WID conform to IMS service architecture presented in 2.3, “Services in IMS” on page 38 and implements service components that are realized using Service Component Architecture (SCA) and packaged as modules. Figure 6-8 on page 131 shows a typical IMS application and its components.

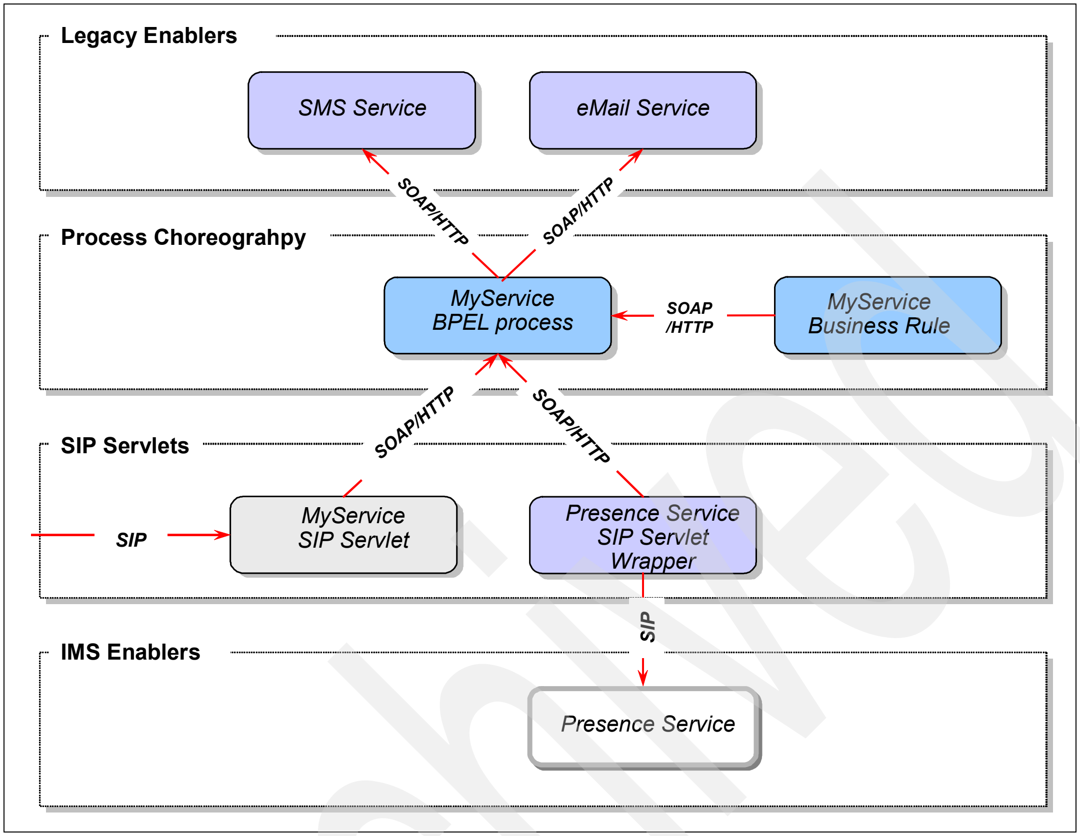


Figure 6-8 Typical IMS application components

The typical IMS application includes the following:

- Exactly one SIP servlet that triggers the BPEL process by invoking the exposed Web Services. Figure 6-9 on page 132 is the illustration of the typical IMS application. It shows a single SIP servlet “MyService SIP” servlet.

> **Note:** The illustration in Figure 6-9 on page 132 shows the SIP servlet “MyService SIP” servlet, BPEL process “MyService BPEL” and imported Web Services

- Optionally, there is exactly one BPEL process (MyService BPEL process) that choreographs existing and IMS enablers.
- The BPEL process may import one or more Web Services from existing enablers (SMS Service, eMail Service).

- One or more Web Services from IMS enablers (Presence Service SIP Servlet Wrapper) may be imported by the BPEL process
- The BPEL process may interact (import services) with components in other service modules such as Business Rules, Human Tasks or Selectors.

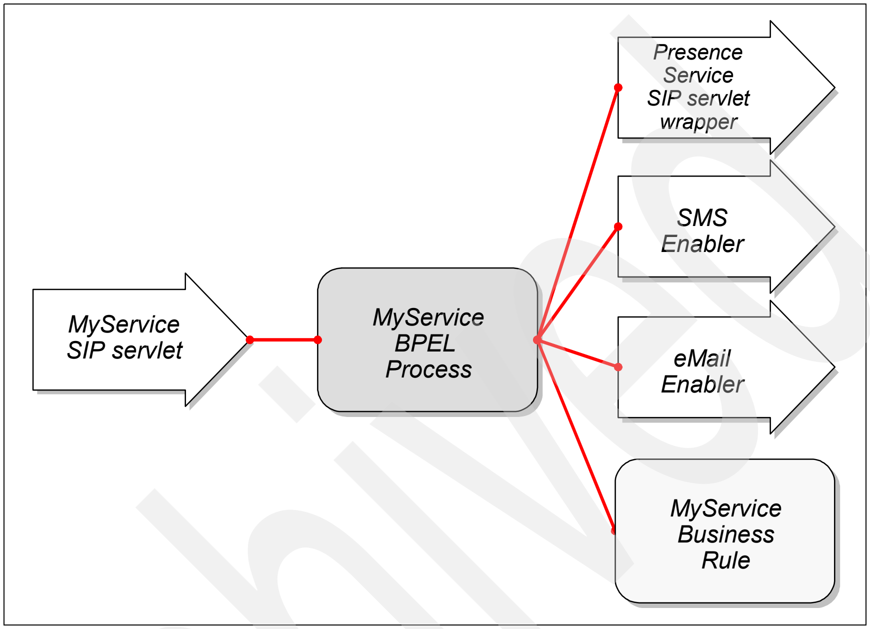


Figure 6-9 Illustration of a typical IMS application

Figure 6-9 shows the WebSphere Integration Developer assembly diagram of the typical IMS application.

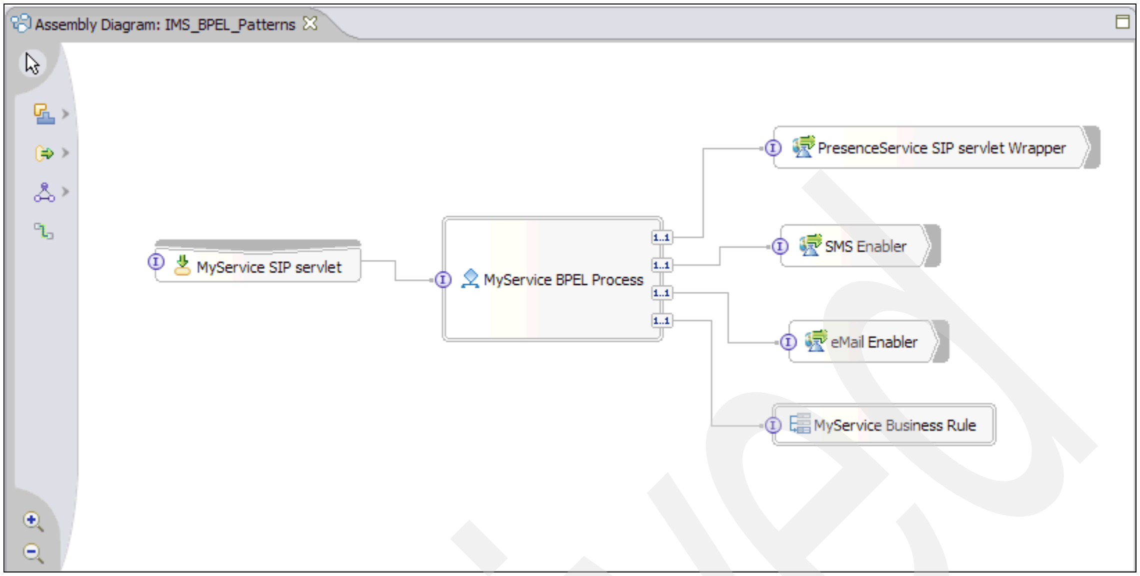


Figure 6-10 Assembly diagram of typical IMS application

6.4.1 Assembling components

There are specific steps that you have to follow to assemble components in order to build service modules for IMS applications. The proposed sequence is just one approach to get you from start to finish. You may change the sequence within certain limits, for example you can take a top-down approach by starting from the assembly diagram.

1. Start by creating your implementation component, in this instance the BPEL process. At this point you don't have to specify the process flow or any other internal logic. You just need to know the component type.
2. Create the business objects that are exposed in the export interface of your component.
3. Create the export interface of your component, its operations and parameters.
4. Import any WSDL files for the enabling or foundation services you want to choreograph with the BPEL process.
5. Create the process flow of the BPEL process.
6. Create the assembly diagram.
7. Test you module.

A detailed example for each of these steps is described in 12.3, “BPEL development” on page 361.

6.4.2 Component tests

WID provides an integrated test environment which enable comprehensive debugging and testing of the modules and components you develop. The environment incorporates integrated versions of the WebSphere Process Server and the WebSphere Enterprise Bus which provides the BPEL and mediation flow runtime environments.

WID provides three tools that work in the integrated test environment to enable you to test, debug and monitor the components and modules that you develop:

- Integration test client

  You can use the integration test client to test your modules and components and report the result of your tests. All testing is performed on the operations defined by the interfaces of your components. The test enable you to determine whether the components are properly implemented and that references are correctly wired. Integration test client supports automatic emulation for unimplemented components and unwired references, this means that your modules need not be fully implemented before you can initiate testing.

  Integration test client intercepts invocations to emulated components or references and routes the control to the associated emulators. Two types of emulators are supported:

  – Manual

    With a manual emulator, you specify the response values for an emulated component or reference at runtime. When a manual emulator is encountered during a test, a manual Emulate event is generated and the test pauses to enable you to manually specify some output parameter values or throw an exception for the emulated component or reference.

  – Programmatic

    In contrast, when a programmatic emulator is encountered during a test, a programmatic Emulate event is generated and the output parameter values or exception are automatically provided by a Java program contained in a visual snippet or Java snippet.

- Integration debugger

  Using the integration debugger you can debug different types of components including visual snippets, business object data maps, business processes, state machines, mediation flows, business rule sets and decision tables.

The integration debugger enable you to add or remove, enable or disable breakpoints. Breakpoints can be set at specific locations where you want execution to pause so you can examine the component status. Depending on the type of component, it is possible to set breakpoints to pause execution prior invocation of the component and immediately on exit from the component.

Execution of code will stop and cause the integration debugger to be invoked if a breakpoint is encountered and the breakpoint is enabled. A disabled breakpoint on the otherhand, will not stop or cause invocation of the integration debugger.

Table 6-3 shows a list of the components you can set breakpoints on and where they can be set.

Table 6-3 Component breakpoints

Editor	Integration components	Where to set breakpoints
Business process editor	Business processes	Activities, Java snippets
Business state machine editor	State machines	States
Business object mapping editor	Business object data maps	Transformations, Java snippets
Business rule set editor	Rule sets	Rules, templates, conditions, actions
Decision table editor	Decision tables	Conditions, actions, values, terms
Visual snippet editor	Visual snippets	Nodes, custom visual snippets, Java visual snippets
Mediation flow editor	Mediation flows	Mediation primitives, nodes

- Event monitor

  The event monitor is used for generating and monitoring of a wide variety of business integration components and their elements. Using the event monitor you can generate and monitor Common Base Event, business process and human task audit events. Common Base Event events are managed by the Common Event Infrastructure (CEI), whilst the business process and human task audit events are logged in the process choreographer database of WebSphere Process Server.

  The monitorable elements for each editor are listed in Table 6-4 on page 136.

Table 6-4 Monitorable elements

Editor	Monitorable element
Assembly editor (CEI only)	Operation
Business process editor (CEI and Audit Log)	Assign, Compensate, Empty, Flow (Parallel Activities), Invoke, Pick (Receive Choice), Process, Receive, Reply, Rethrow, Scope, Script, Sequence, Staff, Switch (Choice), Template (not shown), Terminate, Throw, Variable, Wait, While (While loop)
Business object mapping editor (CEI only)	Map, Transformation (all kinds)
Business rule group editor (CEI only)	Operation
Business state machine editor (CEI only)	Action, Entry, Exit, Guard, State, State Machine Definition (State Machine), Timer, Transition
Human task editor (CEI and Audit Log)	Escalation, Task, Task Template
Interface mapping editor (CEI only)	Operation Binding, Parameter mediation (all kinds)
Selector editor (CEI only)	Operation

Note: Additional information about WID testing, debugging and monitoring is available at:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wbit.help.debug.doc/topics/ccbreak.html

Using the test tools

You invoke the integration test client from the assembly editor. To test a module, right-click anywhere on the canvas and select **Test Module** as shown in Figure 6-11 on page 137.

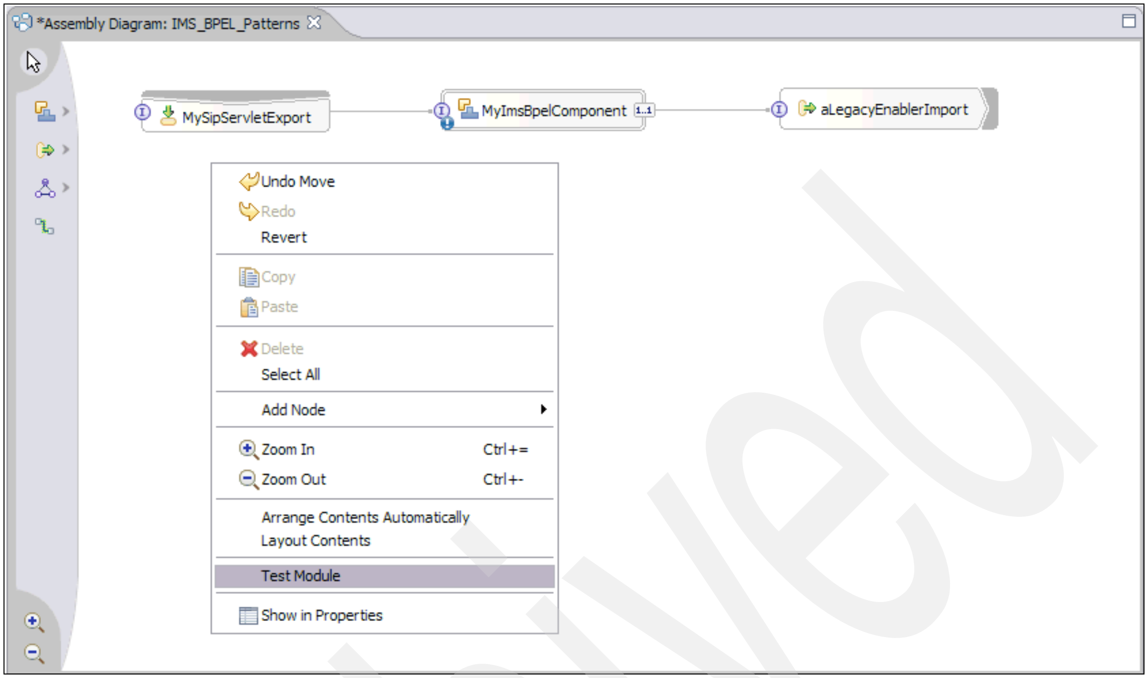


Figure 6-11 Initiating a module test in the Integrated test client

To test a component, select the component, right-click and select **Test Component** as shown in Figure 6-12 on page 138.

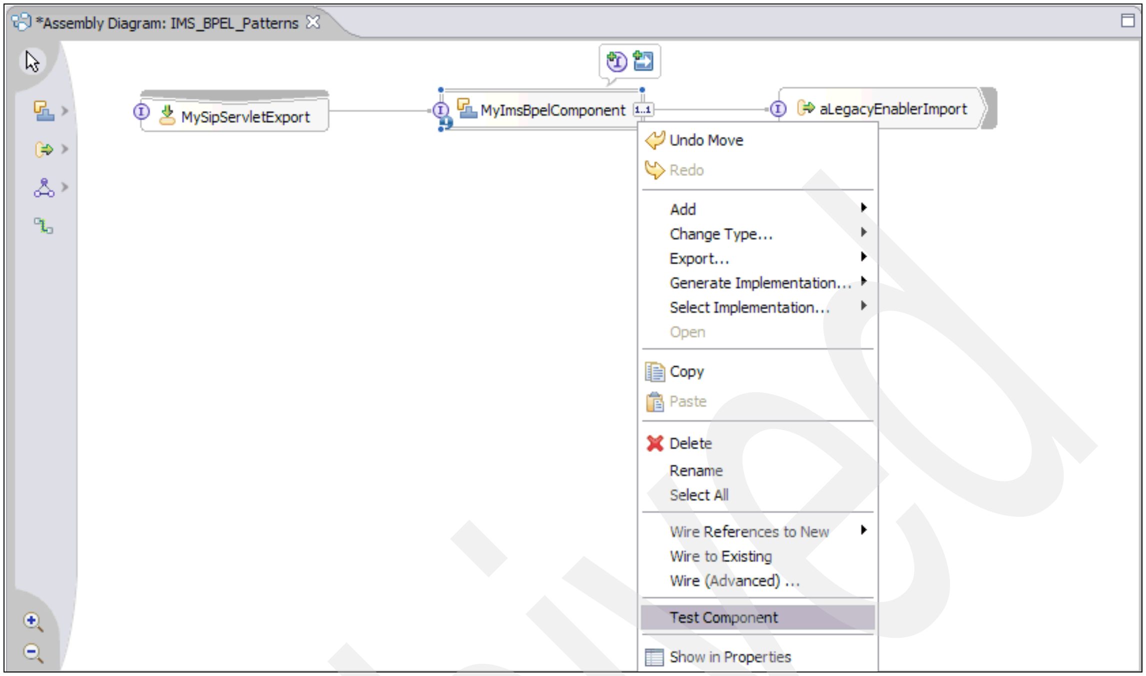


Figure 6-12 Initiating a component test in the Integrated test client

The main interface to configure, start and stop your tests is the Test Editor. Figure 6-13 on page 139 shows an illustration of a Test editor screen.

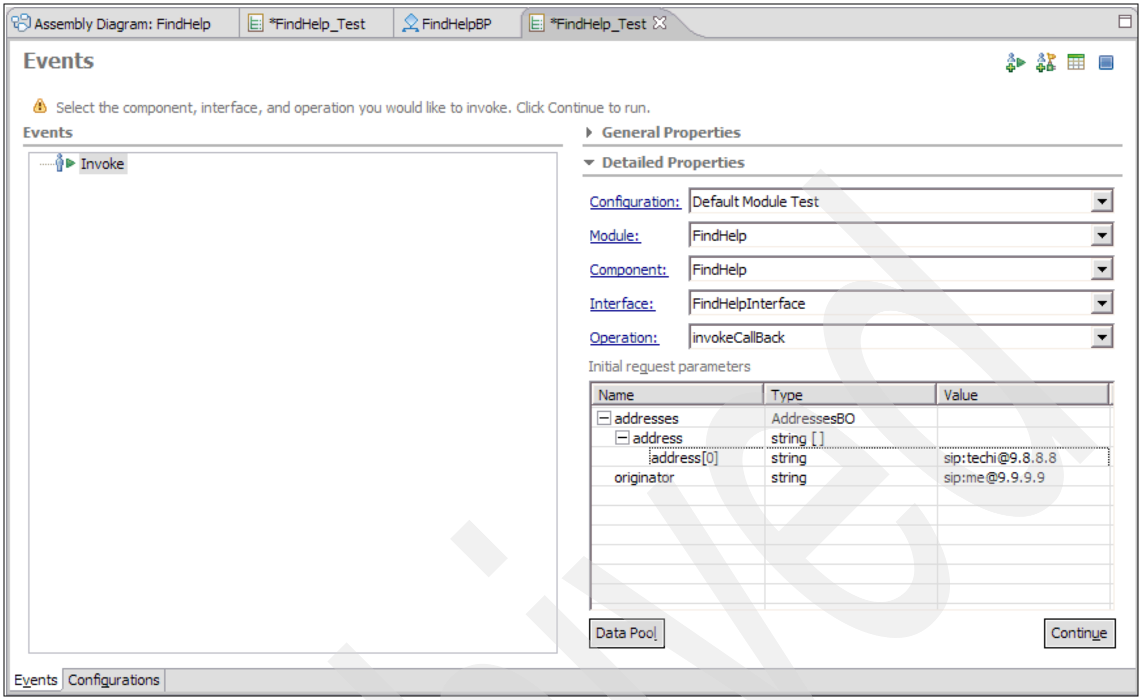


Figure 6-13 Test Editor

The Test Editor consists of two pages:

- Events page

  The events page is the main page, where you specify the test scenario and where you monitor the test execution.

  In the Detailed Properties pane on the right side of the page you define which module, component, interface and operation you want to test. When you have selected an operation, the table Initial request parameters shows all input parameters. You now can assign values to these parameters. For simple types you can enter a value in the column `Value`. For lists and arrays you first need to add a new entry by right-clicking the list parameter and selecting **Add Element**.

  Once you have started the test all monitored events are shown in the Events pane on the left side of the page as shown in Figure 6-14 on page 140.

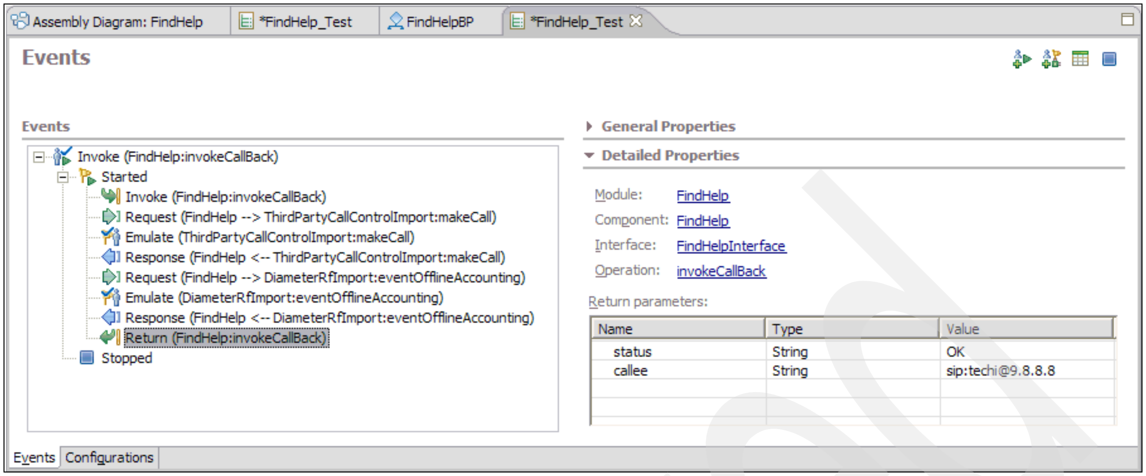


Figure 6-14 Test events

- Configurations page

  The events you monitor during test execution are defined in the Configurations page of the Test Editor. On this page you also specify, which emulators to use as well as the details of the emulation (manual or programmatic). Figure 6-15 shows a sample Configurations page.

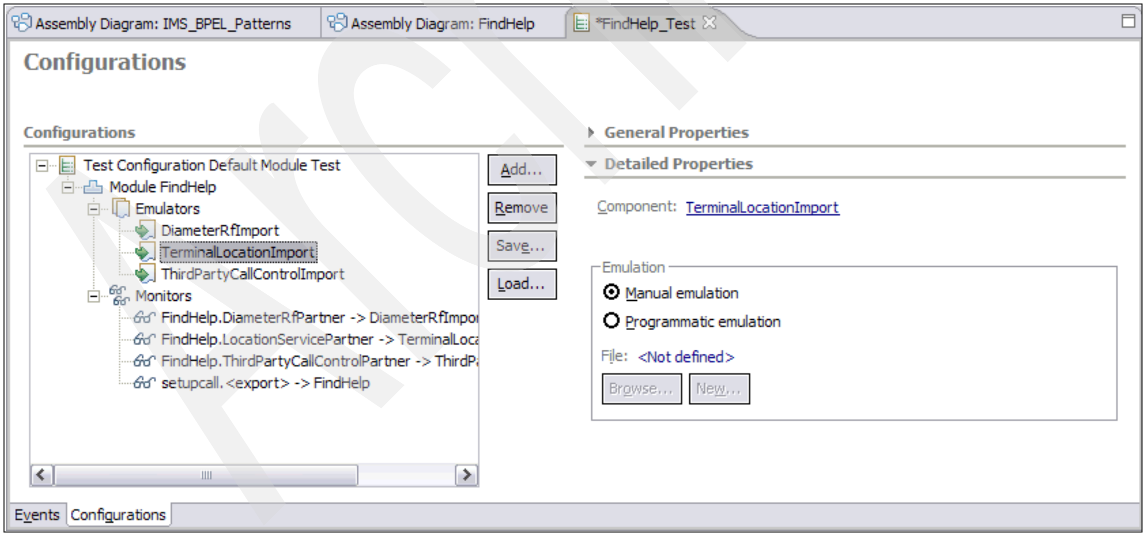


Figure 6-15 Test Configurations page

Using the Add and Remove options, you can add and remove emulators as well as monitors.

Finally a brief test strategy for testing your component:

- First test your component with all imported interfaces emulated.
- Once you core component logic is working to your satisfaction, remove the emulators and thus add the life services one by one.
- Before running a component test with a life external service, run a test of the service itself, to validate if the implementation is up and running and working without failure.

6.5 Technical information

The IBM WebSphere Integration Developer is supported on a number of hardware, software and operating system platforms. The packaging include the IBM Rational Software Development Platform and three components that run on the platform to provide a complete development environment.

6.5.1 Packaging

The WebSphere Integration Developer install image contains four major components:

- The IBM Rational Software Development Platform

  The IBM Rational Software Development Platform is an Eclipse based common development environment that is shared by several products, including:

  - Rational Web Developer
  - Rational Application Developer
  - Rational Software Architect
  - Rational Software Modeler
  - Rational Functional Tester
  - Rational Performance Tester
  - WebSphere Integration Developer

  If you install any of these products, the Rational Software Development Platform is automatically installed as part of the product. If you have more than one of the Rational Software Development Platform products installed, the development platform is installed only once. All of these products have the same user interface, called a workbench, and each product adds functionality to the workbench by contributing plug-ins. A plug-in is a software module that adds function to an existing program or application.

> **Important:** WebSphere Integration Developer 6.0.1 is compatible only with products based on Rational Software Development Platform 6.0.1 (for example, Rational Application Developer 6.0.1). If a different version of Rational Application Developer is detected during the installation of WebSphere Integration Developer 6.0.1, you are required to either upgrade Rational Application Developer to 6.0.1 (available at `http://www.ibm.com/support`) or uninstall your Rational Application Developer so that WebSphere Integration Developer 6.0.1 can be installed successfully.

- WebSphere Integration Developer

  The Eclipse 3.2 based business integration development environment.

- WebSphere Process Server

  An integrated version of the process server. It provides the BPEL runtime environment and enables testing and debugging of the developed processes. You will find more details on this product in 8.9, "WebSphere Process Server" on page 201.

- WebSphere Enterprise Service Bus

  Similar to the BPEL runtime environment, this is an integrated version of the WebSphere Enterprise Bus that provides the runtime environment for mediation flows. It enables testing and debugging of flows. For further details refer to 8.8, "WebSphere Enterprise Service Bus" on page 199.

6.5.2 Supported platforms

WebSphere Integration Developer is supported in the following environments:

Operating systems

- Windows 2000
  - Windows 2000 Advanced Server with SP3 and SP4
  - Windows 2000 Server with SP3 and SP4
  - Windows 2000 Professional with SP3 and SP4
- Windows 2003
  - Windows Server 2003 Enterprise Edition
  - Windows Server 2003 Standard Edition

- Windows XP
  - Windows XP Professional with SP1 and SP2
- Linux
  - Red Hat Enterprise Linux 3.0 WS Update 2
  - SuSE Linux Enterprise Server 9

Hardware requirements

- Intel Pentium III 1 GHz processor minimum (Higher is recommended).
- 1 GB RAM minimum (1 to 2 GB RAM is recommended).
- Disk space:
  - To install the full WebSphere Integration Developer, you will require 5.5 GB of disk space.

    > **Note:** Disk space requirements can be reduced if optional features and run-time environments are not installed.

  - If your file system is FAT32 instead of NTFS, more space will be required.
  - You will require 1 GB in the TEMP directory.
- Minimum display resolution is 1024 x 768 minimum (1280 x 1024 recommended).

7

IBM Telecom Web Services Server Toolkit

This chapter provides an introductory overview of the IBM Telecom Web Services Server Toolkit. The toolkit is a set of additional resources you install in the IBM WebSphere Integration Developer for development of IMS flows.

This chapter contains the following:

- Introduction
- The IBM Telecom Web Services Server Toolkit

7.1 Introduction

The Telecom Web Services Server (TWSS) is one of the major components of the IBM WebSphere Platform for Telecom. It enables the exposure of network and service capabilities through language and technology independent high-level Web service interfaces. The Web service interfaces can be accessed through SIP or Diameter, PSTN functionality through a Parlay or OSA gateway, direct connect access to network protocols, or custom integrated services.

TWSS consists of the following:

- Telecom Web Services service implementations

  Telecom Web Services service implementations consists of several reusable components that are deployed atop the WebSphere Application Server. The components provide high-level service interfaces and the implementations that expose network services for third-party access as Web service interfaces.

- Telecom Web Services Access Gateway

  Telecom Web Services Access Gateway provides policy-driven traffic monitoring, message capture, authorization, and management capabilities. It acts as an intermediary between clients and service end points, by enforcing set policies on all Web service requests and responses that pass through.

  Telecom Web Services Access Gateway extends WebSphere ESB by adding policy-driven processing elements, and includes a number of components called mediation primitives that can be assembled into customized message process flows.

  The mediation primitive interface and programming model provide the points for extensibility and for creation of custom functions. You may choose to use the flows or customize them using WID.

- Service Policy Manager

  Service Policy Manager provides storage capability, access mechanism and administration interfaces for definition, data administration and access to service policies. Using the Service Policy Manager can define requesters and services and attach policies to each. You can also define subscriptions which associate services to requesters.

Note: You can get more information about Telecom Web Services Server from TWSS Information Center at:

`http://publib.boulder.ibm.com/infocenter/wtelecom/v6r1/index.jsp?topic=/com.ibm.twss.intro.doc/esb_extensions_c.html`

7.1.1 Mediation services

Mediation is a new function of the Enterprise Service Bus that allows for the processing of messages between service requesters and service providers. Mediation service applications are implemented using mediation modules which intercept and modify messages that are passed between requesters and providers. Mediation modules contain mediation flows which provide the logic that processes the messages.

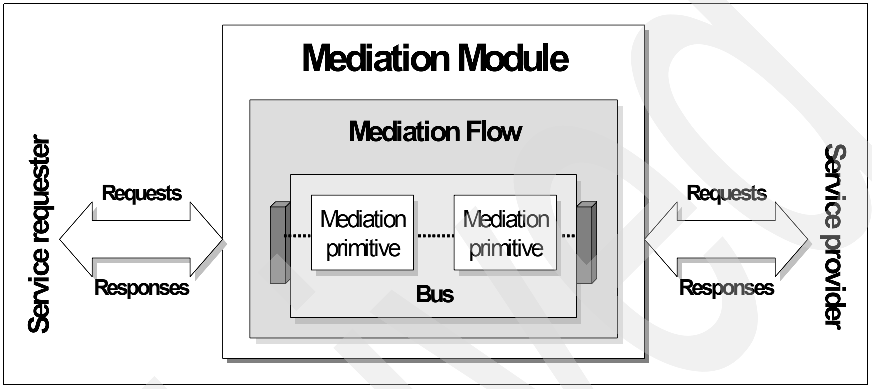


Figure 7-1 Mediation application service

Mediation flows are created and maintained using the Mediation Flow editor. A flow consist of a series of processing steps which are executed in sequence. Using the Mediation Flow editor you define the end nodes of the flow based on the operation connections or source operation. You then add mediation primitives which will be executed in sequence between the end nodes to create request and response flows that provide the processing logic.

Mediation primitives receive messages, process them and propagate the processed messages to the next primitive or node in the flow. Ready-made mediation primitives are available from the Mediation Flow editor palette. You can create custom mediation primitives which provide custom functions not provided by the custom mediation primitives.

Note: For more information about how to configure and develop mediation primitives, see the IBM Redbook *Getting started with WebSphere Enterprise Service Bus V6*, SG24-7212, at:

http://www.redbooks.ibm.com/abstracts/sg247212.html?Open

Also see the IBM WebSphere Developer Technical Journal article, “A practical introduction to message mediation - Part 1”, for the basics of message mediation at:

http://www-128.ibm.com/developerworks/websphere/techjournal/0504_murphy/0504_murphy.html

7.1.2 TWSS Mediation primitives

Telecom Web Services Access Gateway includes a number of mediation primitives. The mediation primitives are divided into two sets:

- Mandatory mediation primitives

  These mediation primitives form the base of Telecom Web Services Access Gateway configuration and flow. They provide base services that support add-on mediation primitive function.

  – Transaction recorder mediation primitive

    The transaction recorder mediation primitive records information about the transaction within a table that is typically referenced by other mediation primitives.

  – Policy/Subscription mediation primitive

    The policy/subscription mediation primitive retrieves policy data based on the requester, service, and operation being called. The data is used as decision parameter in the mediation primitive’s execution.

  – Service invocation mediation primitive

    The Service Invocation mediation primitive extracts the appropriate endpoint from the message and prepares the message further for dynamic service invocation which occurs at the next step.

- Optional mediation primitives

  The following components are optional plug-ins and are used by the default Telecom Web Services Access Gateway flow:

  – Network statistics mediation primitive

    Records message entry and exit information stores the results in a database for use by network operations

– Service authorization mediation primitive

  Provides fine-grained authorization for Web service operations.

– SLA enforcement mediation primitive

  Measures the system use by requestor to enforce policy-driven service level agreements.

– Group Resolution mediation primitive (Parlay X-specific)

  This mediation primitive is used for Parlay X service implementations that can accept group URIs within a list of targets for a given operation. The group resolution mediation primitive expands and replaces group URIs with their member URIs. This mediation primitive is the only optional mediation primitive in the default mediation flow, and is only invoked if group information exists.

– CEI event emitter mediation primitive

  Used by the Fault and Alarm common component to emit a common base event (CBE). By emitting CBEs, the fault information can be picked up by external security or monitoring systems.

7.1.3 TWSS default message flow

Telecom Web Services Access Gateway provides a default flow implementation. The default flow is a model of a typical message processing function by a service provider to support accounting of requests, service/operation level authorization, message capture for regulatory purposes, and traffic level enforcement. The default flow is illustrated in Figure 7-2 on page 150, it shows the mediation primitives wired into the mediation flow.

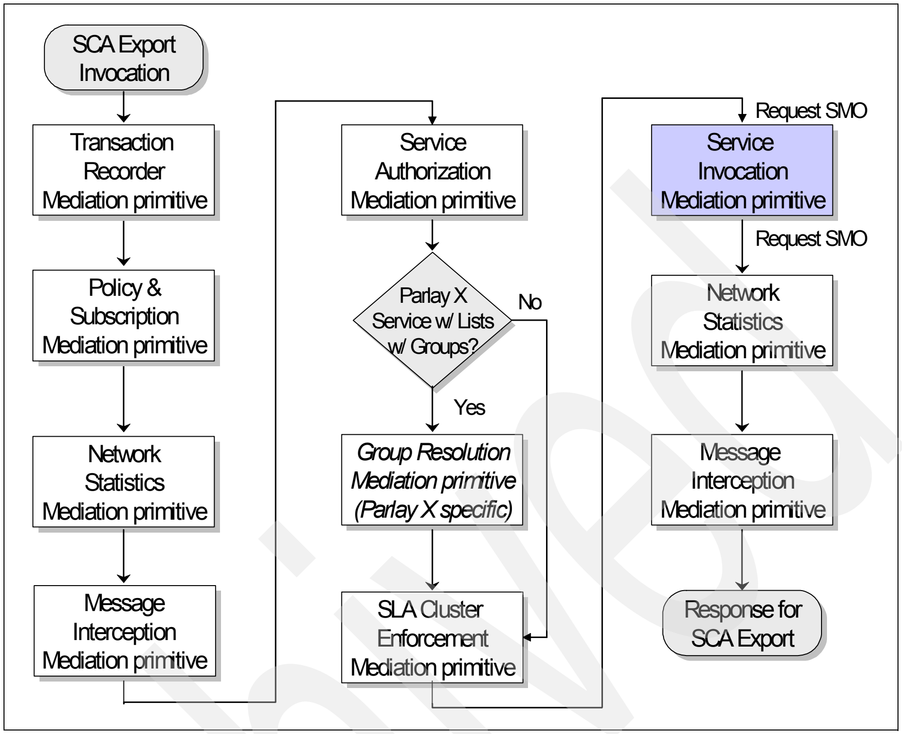


Figure 7-2 TWSS default message flow

7.2 The IBM Telecom Web Services Server Toolkit

The IBM WebSphere Telecom Web Services Server Toolkit consist of TWSS plug-in and ESB Mediation Flows which you install in WebSphere Integration Developer (WID).

Using the toolkit in WID, you can create custom mediation primitives that can be made available in the WID tools palette, assemble flows by dragging and dropping mediation primitives into the Mediation Flow editor and wiring them together and can also customize existing flows using mediation primitives.

A.5, “Installing the Telecom Web Services Server plug-in” on page 524 describes how to install the IBM WebSphere Telecom Web Services Server Toolkit in WID to setup your development environment.

7.2.1 Importing TWSS mediation flows

Having installed the TWSS plug-in IBM WebSphere Telecom Web Services Server Toolkit (TWSS plug-in and ESB Mediation Flows), you need to import the mediation flows you want to work with inside WID. We are going to load the Third Party Call Control flow as an example.

> **Note:** Third Party Call Control Web Service is used in our sample IMS application "`Find Help`" in Chapter 12, "Implementing the IMS sample service" on page 341.

1. To install the Third Party Call Control flow, first start WID from the command line with the clean option.
2. Position yourself in the root directory of your WID installation and type: **`wid.exe -clean`**
3. Once WID is started, it may take quite a while during this cleaning process, even on a powerful workstation (ensure the CPU activity is still going on).
4. Now we are going to import `PX_ESB_TPC_FLOW.zip` that we copied in <WID_install_Root>\runtimes\bi_v6\TWSS\ESB during the installation process (see A.5, "Installing the Telecom Web Services Server plug-in" on page 524).
5. Click **File** → **Import Project Interchange,** select `PX_ESB_TPC_FLOW.zip` and click **Open**.

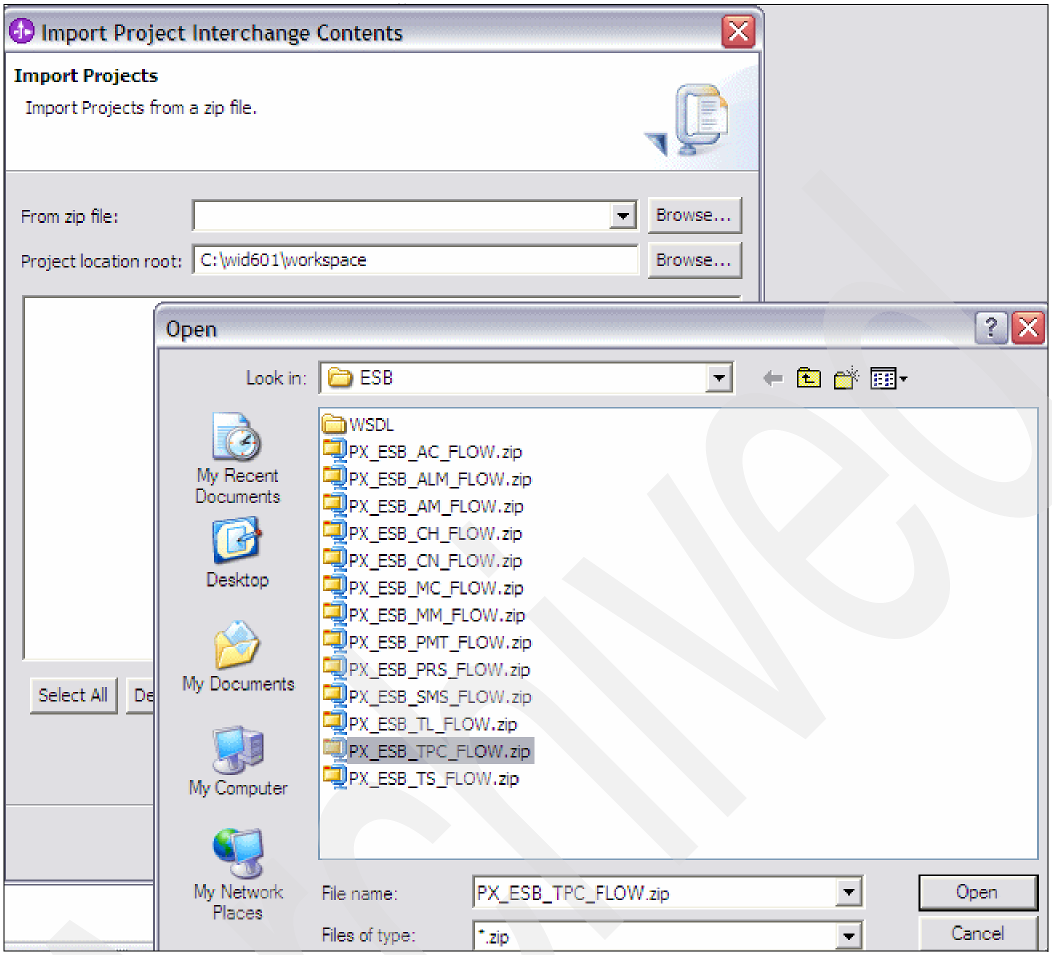


Figure 7-3 Import Parlay X Mediation Flow

6. In the next window, check the box for the selected flow and click **Finish**.

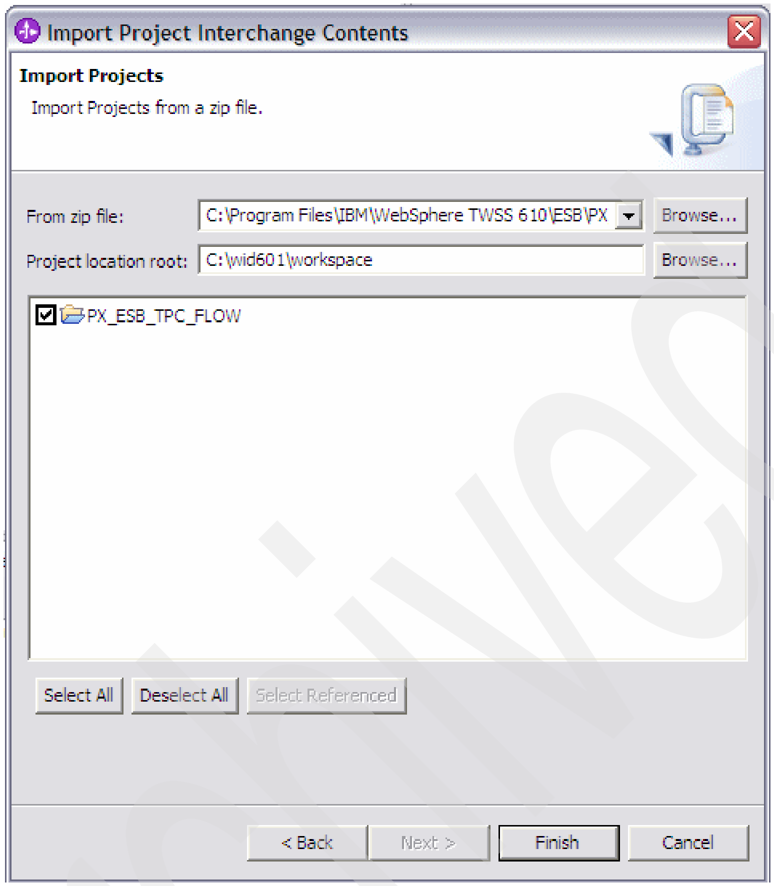


Figure 7-4 Select 3 Party Call Control flow

7. You can now expand the **PX_ESB_TPC_FLOW** folder as well as the **Mediation Logic** folder.
8. Double-click **PX_ESB_TPC_FLOW**.
9. In the **Mediation Flow Editor** that appears on the right, resize the top and bottom panes to approximately same size.
10. Click the **makeCall** operation, the first operation in **ThirdPartyCall** interface box.
11. The panes should look like the image in Figure 7-5 on page 154.

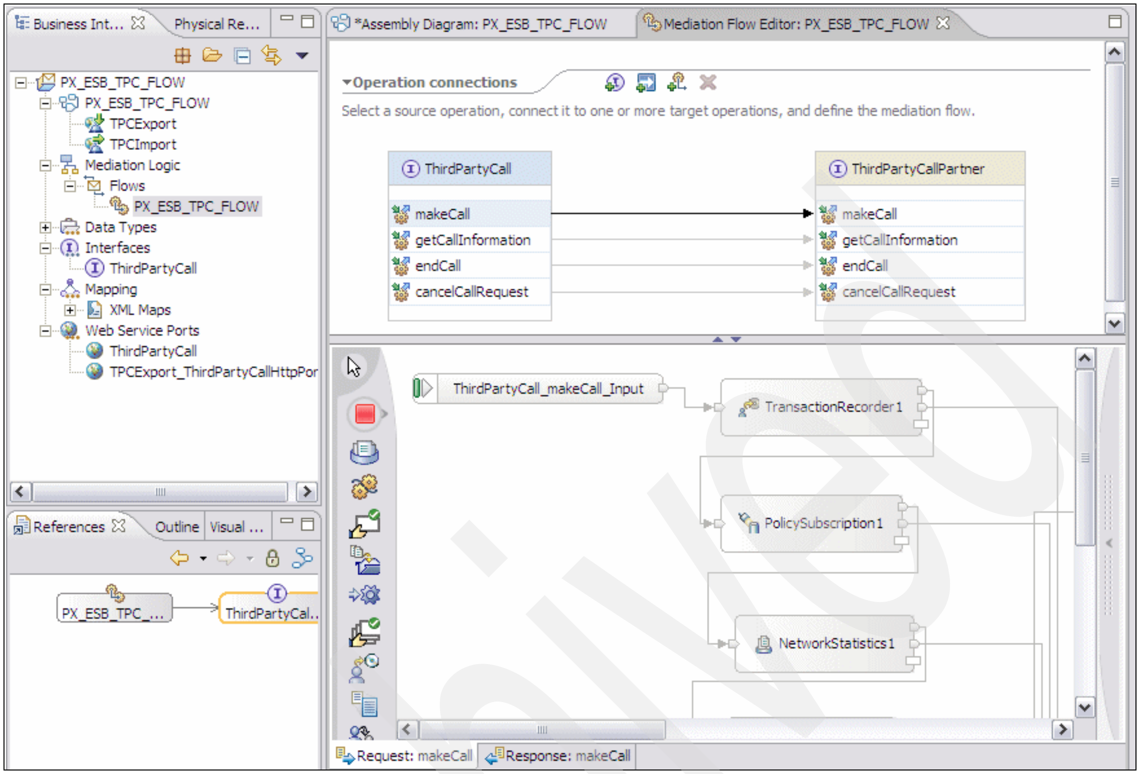


Figure 7-5 Third Party Call Control flow in the Mediation Flow editor

7.2.2 Working with TWSS mediation flows

With the mediation flow loaded you are ready to start working with it.

- You start by double-clicking the title tab for the **Mediation Flow Editor** window to get an enlarged view of the flow.

> **Note:** You can observe that the mediation flow has a similar structure as the default TWSS default flow in Figure 7-2 on page 150.
>
> You can also observe that the TWSS Toolkit mediation primitives are available for addition in the flow.

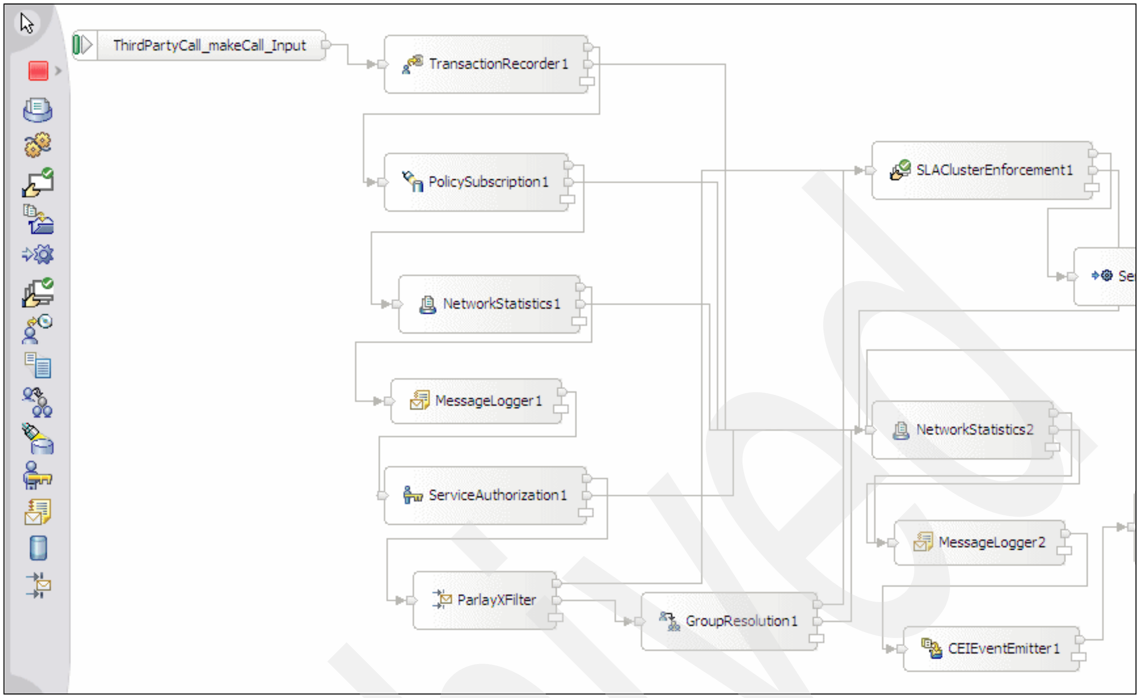


Figure 7-6 Third Party Call Mediation Flow

- Expand the bottom pane of the window where the properties of the selected object is displayed.
- Select **TransactionRecorder1**
- The Description and Details in the Properties pane should appear as in Figure 7-7 on page 156 and Figure 7-8 on page 156.

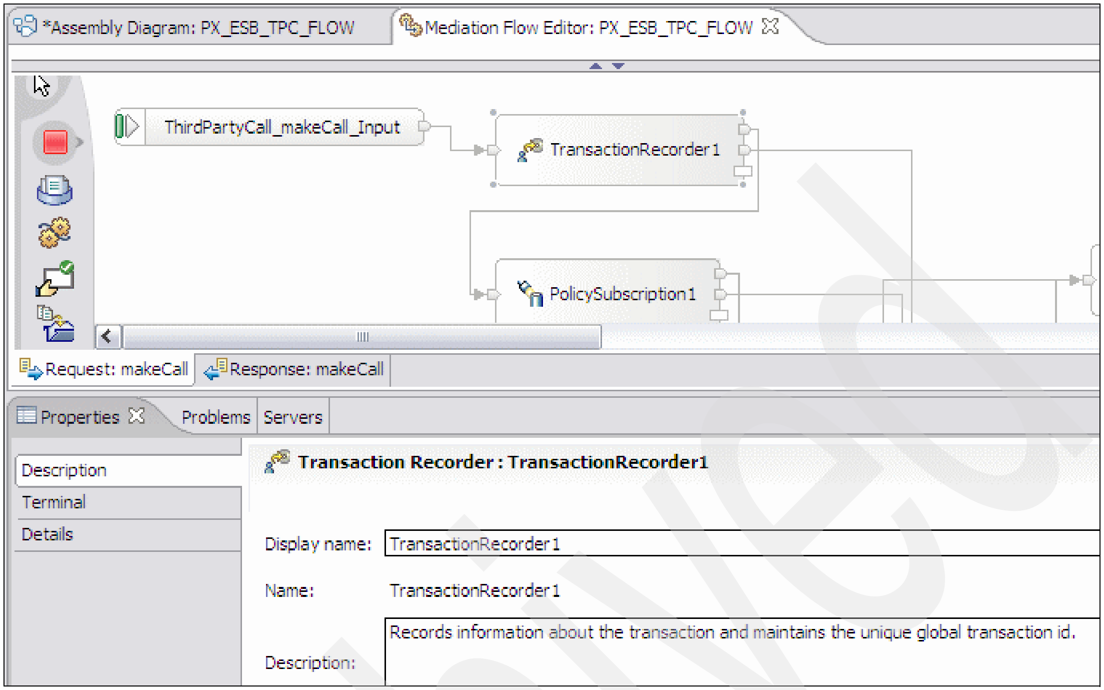


Figure 7-7 Description of TransactionRecorder1 mediator

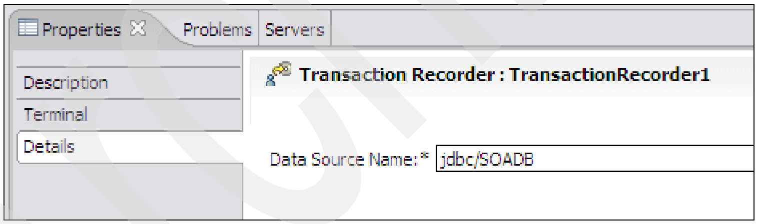


Figure 7-8 Data source detail of TransactionRecorder1 mediator

> **Note:** It is also possible to see information similar to the one provided in **Properties** window by positioning the cursor above the TransactionRecorder1 mediation box. By having the cursor hover over mediation primitives in the editor, a pop-up will display information about the mediation.

You can wire the mediation primitives together by clicking the out connection of the source node and connecting it to the destination. Figure 7-9 on page 157 shows the illustration of wiring **TransactionRecorder1** to **PolicySubscription1**. It also shows information in the **Terminal** tab of the **Property** pane.

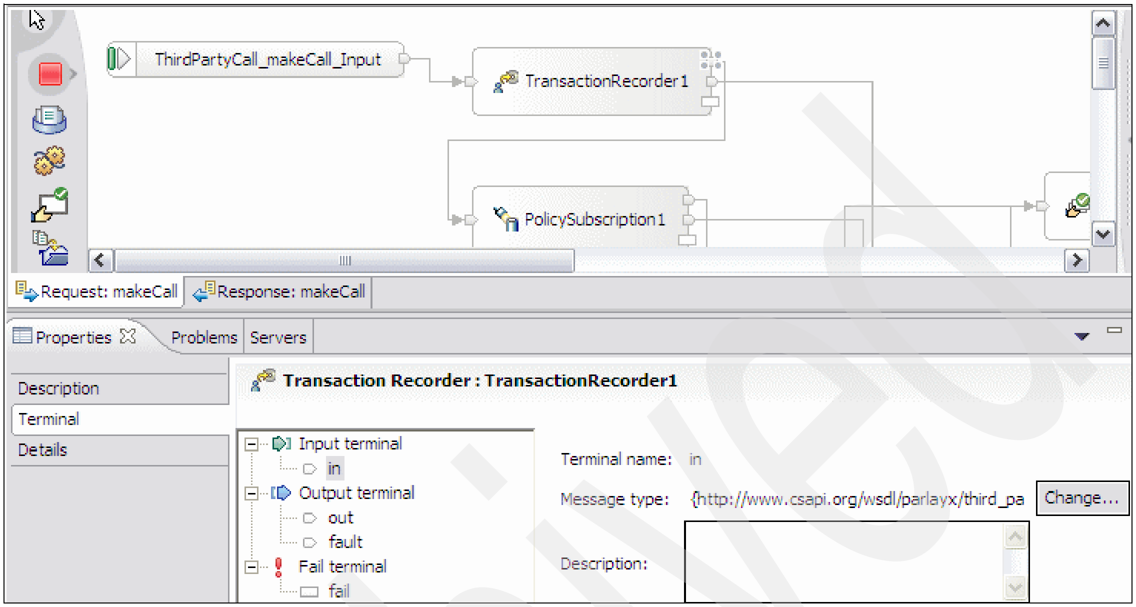


Figure 7-9 Wiring TransactionRecorder1 to PolicySubscription1

You can use the editor to modify this mediation flow or any other mediation flow by adding new mediation primitives and rewiring the connections to change the flows. In 12.3, “BPEL development” on page 361 you can find the sample IMS application “`Find Help`” which shows how to modify mediation flows.

8

Introduction to the IBM service execution environment

The IBM IMS solution focuses on the service development, deployment and execution environments. In this chapter we introduce the converged service execution environment where the services you develop using the different toolkits that are described in this redbook will run.

This chapter contains the following:

- Overview of the IBM IMS solution
- The IBM WebSphere Application Server
- WebSphere IMS Connector
- WebSphere Presence Server
- Telecom Web Services Server
- WebSphere Enterprise Service Bus
- WebSphere Process Server

8.1 Overview of the IBM IMS solution

In 2.2.4, “Functional planes” on page 36 we identified the functional planes of the 3GPP architecture. The planes include:

- The transport plane

  It provides access network control for devices attached to the network.

- The control plane

  The control plane provides the traffic routing capabilities for the IMS network. It directs traffic towards the service plane for the invocation of services and towards the transport plane to establish connections and direct session resource usage.

- The service plane

  This plane contains the application platforms that host service logic and provide integration between IMS and non-IMS services.

- The management domain

  The management domain provides supporting services to the different planes, including Operational Support Systems (OSS), Business Support Systems (BSS), Systems Management Services (SMS) capabilities.

- The service creation domain

  The service creation domain populates the service plane and appropriate systems within the management domain with the service logic and configuration data required to operate and manage services.

The IMS logical layers define the typical infrastructure on which solutions are deployed. The IBM IMS solution architecture consists of full life cycle service delivery environments that focus on the service plane, service creation and management domains.

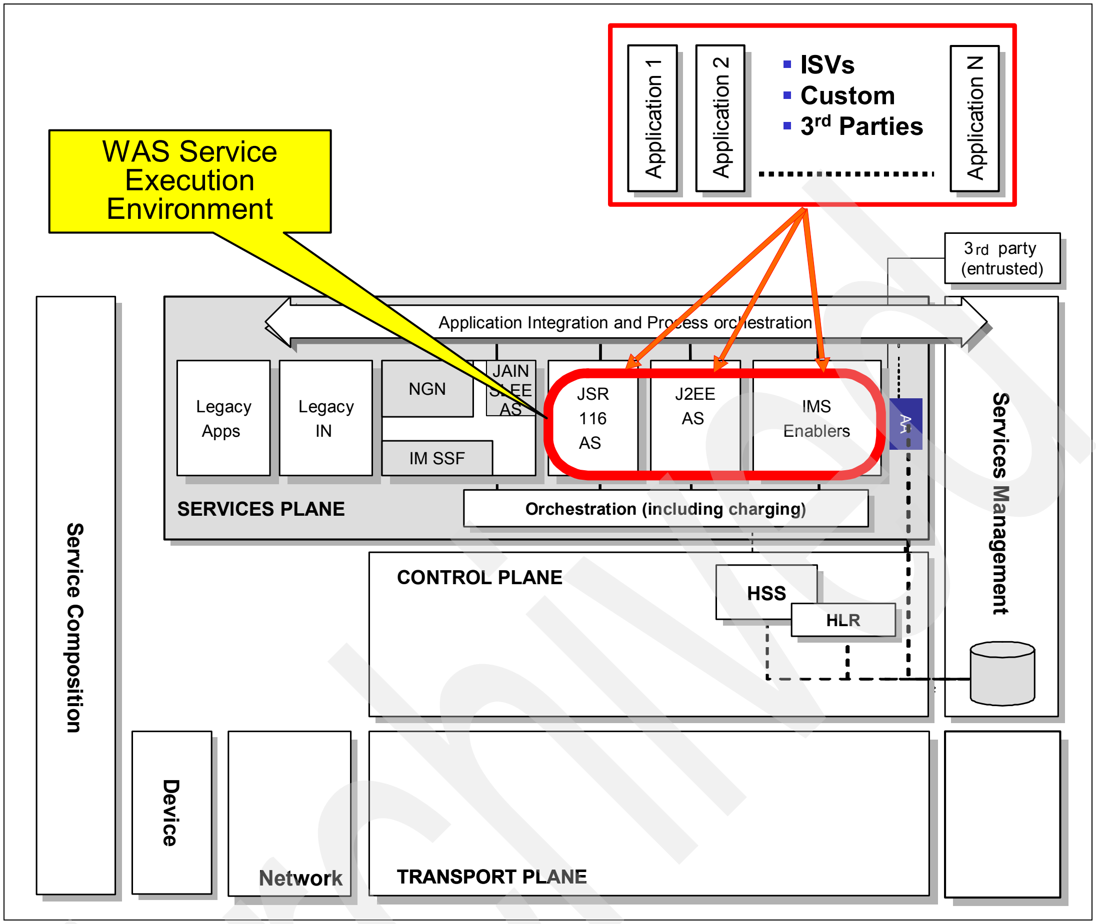


Figure 8-1 IMS service delivery environment

The IBM IP Multimedia Subsystem offering includes integrated hardware and software components that provide a flexible, high-performance IMS-compliant service delivery platform. The platform includes:

- Service development environment

  The IBM Unified Service Creation Environment provides the development platform and processes that decreases time-to-market between the conception of a user-service and its deployment, possibly involving multiple service execution environments.

- Service deployment environment

  Deploys the user-services on their execution environments. It provides capabilities for testing and prototyping user-services before deployment.

- Service delivery environment

  The service delivery environment consists of middleware and hardware systems for hosting and managing converged services combining voice, video and data over both fixed and mobile networks and leveraging service-oriented architecture (SOA).The service delivery environment itself consists of four sub environments which include service execution, networks, user devices, and service management.

8.1.1 The service execution environment

The IBM service execution environment is built on the latest release of IBM WebSphere Application Server, which has deeply integrated SIP technology to deliver a truly converged HTTP/SIP service execution platform for next generation services. It provides full support for JSR116 and other pertinent RFCs. The service execution environment also delivers prebuilt IMS-compliant service enablers that include Presence, Group List Management, Diameter and IMS Session Control (ISC) interfaces, as well as Parlay X interfaces for security-rich, policy-based third-party access to network elements.

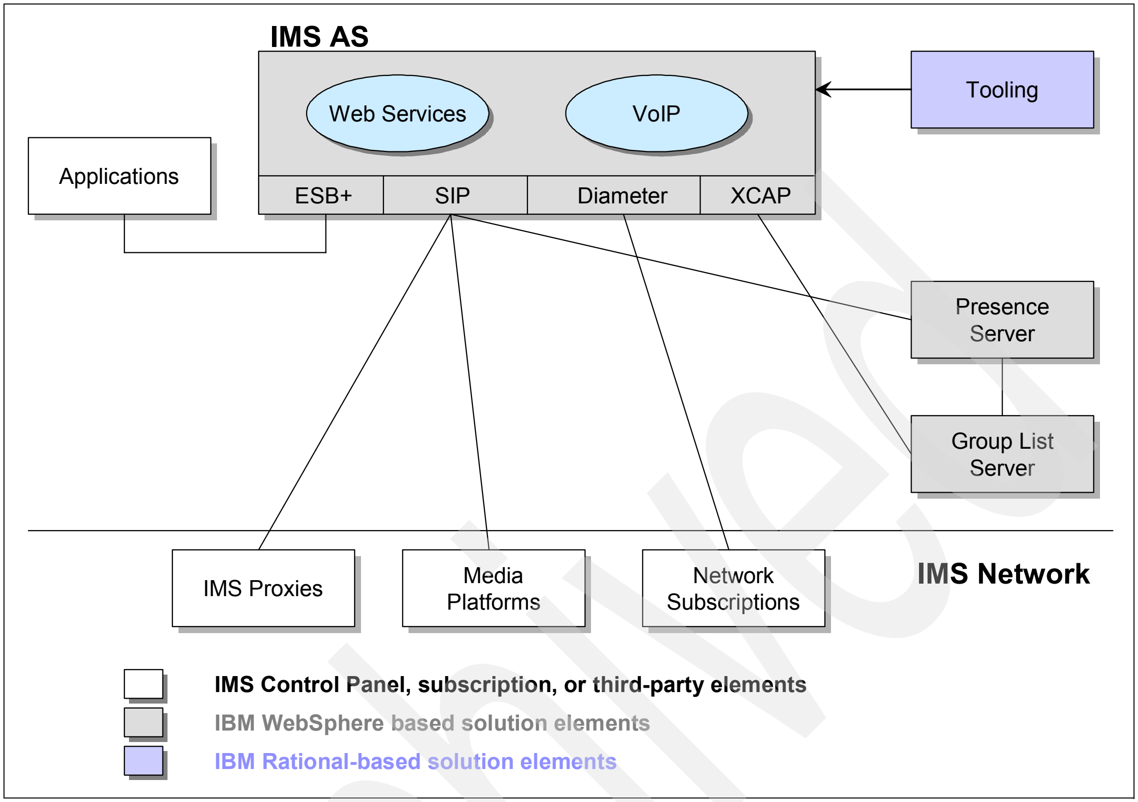


Figure 8-2 The IBM IMS solution overview

The fully converged SIP/HTTP environment is part of the IBM next generation service platform. It executes SIP-based, IMS services and non-IMS services and delivers the same services to both wireline and mobile networks.

Services can be built from standard infrastructure such as JSR 116 SIP Servlet or using other modular services enablers as illustrated in Figure 8-3 on page 164.

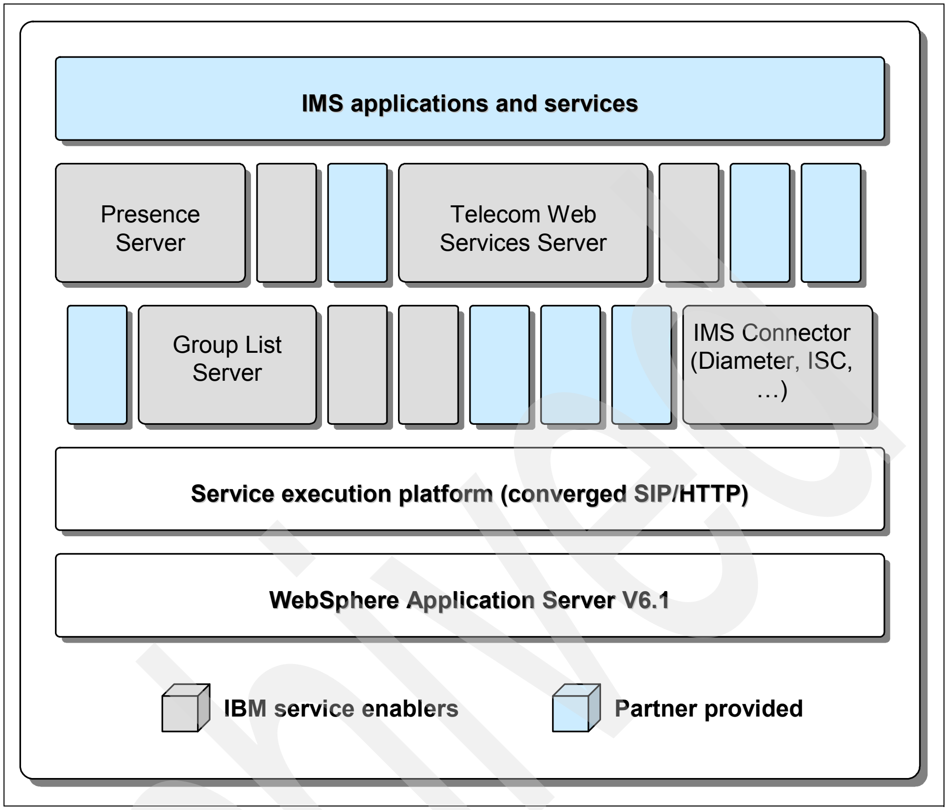


Figure 8-3 IBM IMS service execution environment

IBM delivers the following modular service enablers for the development and delivery of next generation services:

- IBM WebSphere IMS Connector

  It enables applications running on WebSphere Application Server to communicate with IMS core elements like the Call/Session Control Function (CSCF) and Home Subscriber Servers (HSS), as well as billing systems. It includes IMS Service Control (ISC) and Diameter client interfaces.

- IBM WebSphere Presence Server

  It also includes a Group List Server. The presence server delivers the ability to collect, manage and distribute real-time information about subscriber access, availability and willingness to communicate across applications and environments and to create and manage application-independent, network-stored groups.

- IBM WebSphere Telecom Web Services Server

  It delivers a security enhanced, standards-based Web Services gateway for third-party access to network capabilities like presence and call control.
- The WebSphere Enterprise Service Bus and WebSphere Process Server

  The IBM IMS solution integrates services by making use of service-oriented architecture (SOA). The WebSphere Enterprise Service Bus and WebSphere Process Server provide the runtime infrastructure SOA.

The modular service enablers are described in more detail in this chapter.

> **Note:** IBM is actively engaged in interoperability testing with major Network Equipment Provider (NEP) partners. IBM is also working with its telecommunication-related business partners on interoperability to ensure complete IMS solutions including IMS services plane, control plane elements and value-added services.

8.2 The IBM WebSphere Application Server

WebSphere Application Server V6.1 is the foundation of the IBM WebSphere software platform, and a key building block for service-oriented architecture (SOA). As a leading J2EE 1.4 and Web Services application platform, WebSphere Application Server V6.1 delivers a high performance transaction engine you can use to build, run, integrate, and manage dynamic applications. WebSphere Application Server V6.1 includes a number of new and enhanced features. The most significant feature is the implementation of JSR 116, which standardizes servlets that consume and produce SIP signaling interactions.

> **Note:** The following resources provide an overview of the IBM WebSphere Application Server V6.1 and J2EE the Java based programming model for enterprise applications:
>
> - *Experience J2EE! Using WebSphere Application Server V6.1*, SG24-7297
> - *WebSphere Application Server V6.1: Technical Overview*, REDP-4191

The highlights of the new features implemented in WebSphere Application Server V6.1 include the following:

- JDK™ 5.0 innovations

  IBM implements J2SE™ 5.0 specification as Java Development Kit 5.0. It introduces significant improvements in programmer productivity, application portability, and performance. IBM innovations in the Virtual Machine enhances modularity, profiling, debugging, serviceability, and efficiencies gained through compile and run-time performance optimizations.

- New automation tools

  WebSphere Application Server V6.1 provides Application Server Toolkit a full-scale Integrated Development Environment. It offers features such as color-coded source display, command completion, configuration navigation, and syntax checking.

- IBM Installation Factory for WebSphere Application Server V6.1

  Uses self-managing autonomic technology to make WebSphere Application Server installation and deployment easy, reliable and repeatable. By streamlining the up-and-running process to just one simple step. New in this release, IBM Installation Factory for WebSphere Application Server V6.1 now allow you to pre-package appropriate service stream levels and to include applications and their configurations. You can now also generate cross-platform install packages, saving you time that can be used to focus on real business issues.

- Tight integration with IBM Rational tools

  Provides a rich, easy-to-use development environment that deploys directly to WebSphere Application Server V6.1. The next version of Rational Application Developer is expected to bundle new Rational toolset and provide tighter integration.

- Application Server Toolkit (AST) enhancements

  The AST provides basic support for the creation of new applications targeting WebSphere Application Server V6.1. This includes wizards and tools for creating new Web applications, Web Services, portlets, and EJBs, as well as annotation based programming support, new administration tools for the creation and maintenance of wsadmin Jython files, and tools to edit WebSphere-specific bindings and extensions. The AST also provides the tools necessary to develop and export JSR 116 SIP Servlets, including wizards to create SIP Servlets, a rich Deployment Descriptor Editor, and import/export wizards to package and deploy SIP Servlets. Support is also included for the rapid deployment feature, providing an easy mechanism to deploy applications to a running V6.1 server. A comprehensive Unit Test Environment is included, providing all function necessary to deploy and debug applications on WebSphere Application Server V6.1. This support includes a Unit Test Client Web application for easy testing of EJBs and Web Services.

- Performance and high availability

  J2SE Development Kit 5 enables WebSphere Application Server V6.1 to deliver substantial performance improvements and to continue to drive industry benchmarks. The proxy server has also been integrated for cache off-loading, which improves application performance and scalability.

- Session Initiation Protocol (SIP) servlet support

  SIP servlet support is integrated within WebSphere Application Server V6.1. With the converged servlet engine, you can easily enable converged HTTP and SIP interactions, providing portlets, HTTP servlets, and SIP Servlets.

- Powerful Web Services

  WebSphere Application Server V6.1 delivers new Web Services standards to more securely extend your reach to new environments. The latest enhancements provide better application portability and control, as well as performance improvements, due to faster parsing technology, SOAP/JMS enhancements, and changes to SOAP with Attachments API for Java (SAAJ).

> **Note:** The Network Deployment (ND) version of the WebSphere Application Server product brings in clustering, proxy servers, Web servers, IP load balancing, and an industry leading high availability service. The Extended Deployment (XD) version includes advanced QoS, management, and monitoring capabilities.

8.2.1 WebSphere Application Server SIP support

The WebSphere Application Server V6.1 implements SIP Servlet 1.0 specification. It includes a SIP stack and a SIP container implementing JSR 116. The HTTP container and SIP container are converged and are able to share session management, security, and other attributes. The integrated implementation provides advantages for SIP application support in WebSphere Application Server. With integrated implementation, applications that include SIP Servlets, HTTP servlets, and portlets can seamlessly interact, regardless of the protocol.

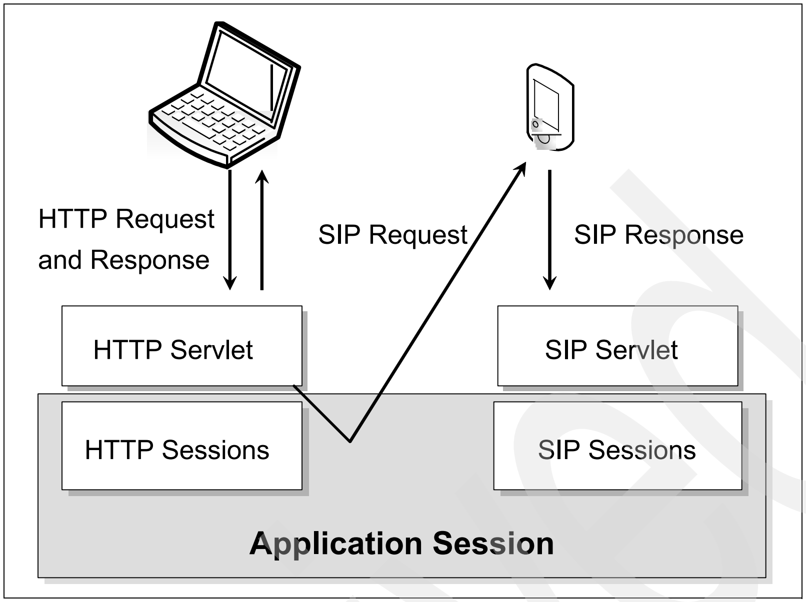


Figure 8-4 Converged HTPP/SIP application session

SIP messages can be transported via UDP (User Datagram Protocol) or TCP and therefore UDP support has been added for SIP. Traffic arriving via TCP will be routed either to the SIP or HTTP component depending on the content of the inbound message. Message are pre-processed before being passed to the Servlet container for routing to the appropriate application(s).

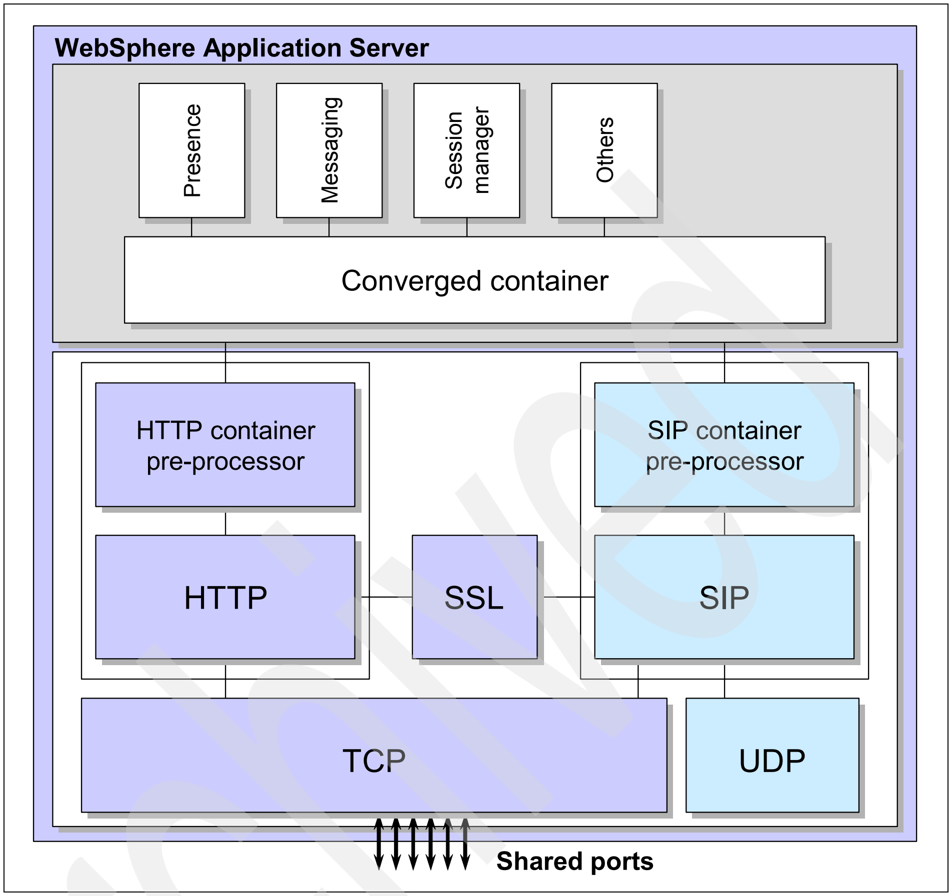


Figure 8-5 Converged container handling HTTP Servlets, Portlets, and SIP Servlets

Compared to a stand-alone SIP container, the converged container provides several benefits including the following:

- Reduced latency between containers

  SIP is a real-time protocol and a timely response to a SIP request is critical. For a combined SIP and HTTP application, any latency between the HTTP and SIP environments must be minimized. A converged SIP HTTP container ensures that the latency between the environments is as optimal as possible.

- Single point of administration

  There is a single point of administration for the SIP and HTTP containers, instead of one administration console for SIP and another for HTTP. It is possible to deploy, in one step, a SAR archive containing an HTTP and a SIP

Servlet. An administrator who is used to WebSphere Application Server, will find it straight forward to administer SIP applications.

- JMX™ and command line support for configuration and administration

  Similar to HTTP administration, SIP can utilize JMX (Java Management Extensions) and command line configuration to administer the system. This allows administrators and developers to automate processes in a common environment for both SIP and HTTP.

- Access to service-oriented architecture

  Integrated access to Web Services, Enterprise Service Bus and Service Orchestration is provided. This allows new services to be created and exposed via an Enterprise Service Bus.

Standards support

WebSphere Application Server V6.1 implements a number of industry wide standards. The following SIP standards are supported in WebSphere Application Server:

Table 8-1 Base SIP standard support

Standards organization	**Standard**	**Title**
IETF		
	RFC 3261	SIP core protocol
	RFC 3262	Reliability of provisional responses in SIP
	RFC 3265	SIP specific event notification
JCP		
	JSR 116	SIP Servlet APIs

Several application level standards are not implemented in WebSphere Application Server V6.1 however, it does not prevent the implementation nor the compliance by applications running in the converged HTTP/SIP container. An example is RFC 2976, where an application may add support for SIP INFO method and the container will not prevent its use. The standards include but are not limited to the following:

Table 8-2 Other supported SIP standards

Standards organization	Document	Title
IETF		
	RFC2976	SIP INFO method
	RFC 3428	SIP extension for instant messaging
	RFC 3455	Private header extensions to SIP
	RFC 3327	Extension (Path) header field for registering nonadjacent contacts
	RFC 3725	Best current practices for third party call control.
	RFC 3856	SIP-Presence event package
	RFC 3863	Presence information data format
	RFC 3680	SIP-Event state publication
	RFC 3903	SIP-Registration event package
	draft-ietf-simple-rpid	Rich Presence Information Data Format
	draft-ietf-simple-event-list	SIP - Subscription on collection of resources
	draft-ietf-sipping-uri-list-subscribe	SIP - Subscription on a list of URIs

Note: See WebSphere Application Server V6.1 compliance with industry SIP standards at:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/rsip_refstandard.html

SIP Proxy and high availability

The WebSphere Application Server V6.1 proxy server delivers a high performance SIP proxy capability that can be used at the edge of the network to route, load balance, and improve response times for SIP dialogs to back-end SIP resources.

The SIP proxy design is based on the WebSphere Application Server HTTP proxy architecture and can be considered a peer to the HTTP proxy. Both the SIP and the HTTP proxies are designed to run within the same WebSphere Application Server proxy server and both rely on filter-based architecture for message processing and routing.

The SIP proxy serves as the initial point of entry, after the firewall, for SIP messages. You can use rules to configure which SIP proxy to route messages to and to load balance clusters of SIP containers.

WebSphere Application Server V6.1 proxy uses the unified clustering framework and high availability manager services of WebSphere Application Server Network Deployment package to seamlessly monitor the health of the servers, handle the workload, routing and to support the session affinity needs of the SIP container.

The Proxy server can be fronted by a simple IP sprayer, such as the Load Balancer component included in WebSphere Application Server Network Deployment. If a Proxy Server fails, the affinity is to the container and not to the proxy itself so there is one less potential failure along the message flow.

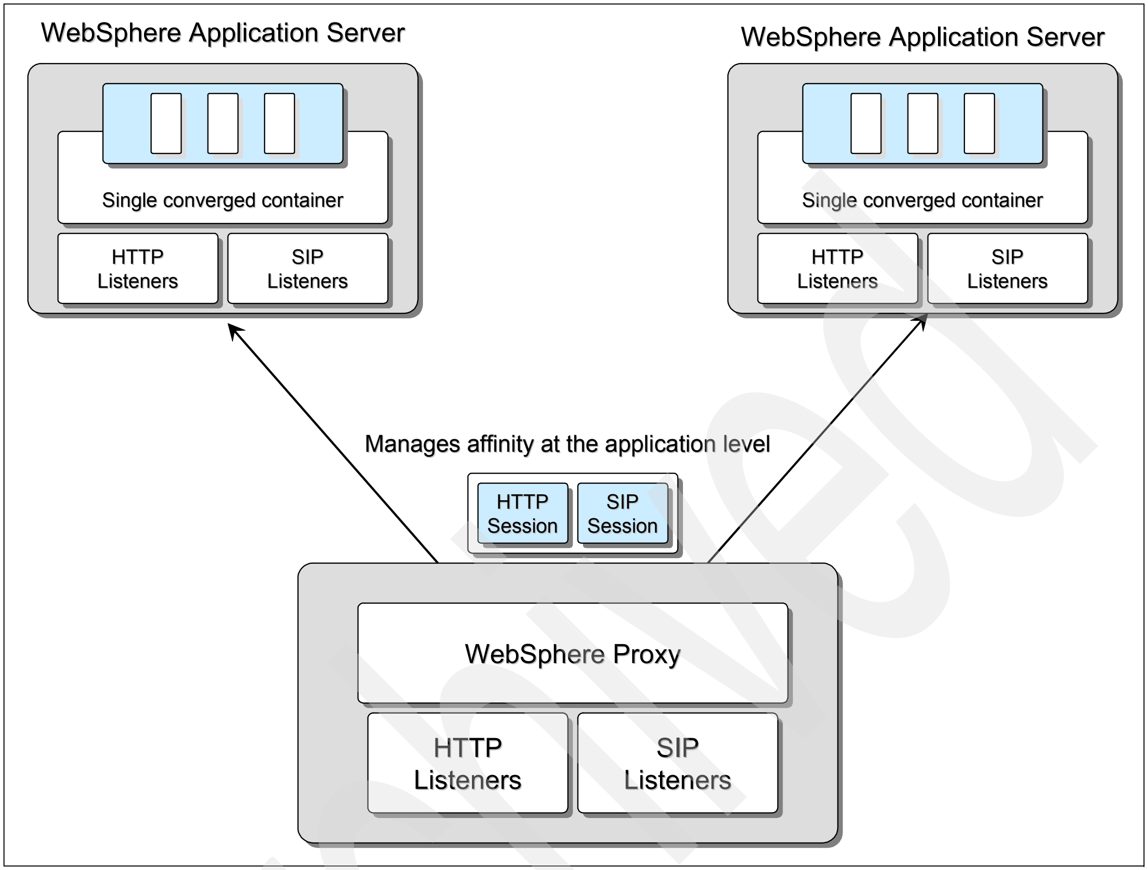


Figure 8-6 SIP and HTTP proxy and workload balancing

8.3 WebSphere IMS Connector

WebSphere IMS Connector adds IMS-specific interfaces to the WebSphere Application Server V6.1 to deliver a full IMS standards-compliant SIP application server. Coupled with the IMS-ready HTTP and SIP applications support in WebSphere Application Server, WebSphere IMS Connector enable truly converged IMS applications to be built and deployed on a single service execution platform.

WebSphere IMS Connector adds two key IMS-compliant interface elements, as defined by the 3rd Generation Partnership Project (3GPP) and Internet Engineering Task Force (IETF) to the WebSphere Application Server platform. These are:

- IMS Session Control (ISC) interface

  ISC enables standards-based connectivity to Call/Session Control Functions (CSCF) in the IMS control plane

- Diameter stack and interfaces

  Support subscriber management and charging functions in accordance with IMS standards

8.3.1 ISC interface

3GPP specifies a standard interface based on SIP for how the IMS Core plane communicates and integrates with Application Servers in the IMS service plane. This interface is called IMS Service Control (ISC).

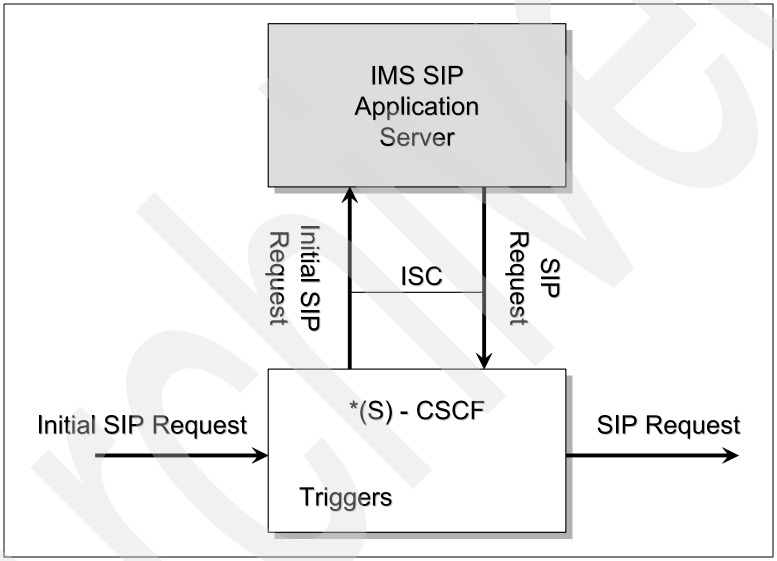


Figure 8-7 ISC Interface

The ISC Interface is a bidirectional interface that uses SIP messaging. It is based on IETF RFC 3261, which specifies the standardized SIP messages exchanged between the CSCF residing in the IMS core and application servers.

The ISC interface is formally defined by 3GPP and 3GPP2 in the following standards:

- 3GPP

  3GPP TS 23.228 Technical Specification Group Services and System Aspects; IP Multimedia Subsystem (IMS); Stage 2 (Release 6)

- 3GPP2

  3GPP2 X.S0013-002 All-IP Core Network Multimedia Domain (MMD); IP Multimedia Subsystem; Stage 2

The role of the ISC Interface with regards to the application server is to provide service invocation and present SIP parameters to applications.

Service Invocation and Interaction

The service platform or device clients trigger initial SIP request at the Service Proxy (Serving CSCF) located in the IMS Core. The CSCF proxies the service request to corresponding application based on triggers. The application server acts either as a user agent (originating or terminating), proxy server, or B2BUA (back-to-back user agent). The application server may Record and Route SIP requests to stay in the signaling path, and the CSCF maintains the states between dialogs sent to or from applications.

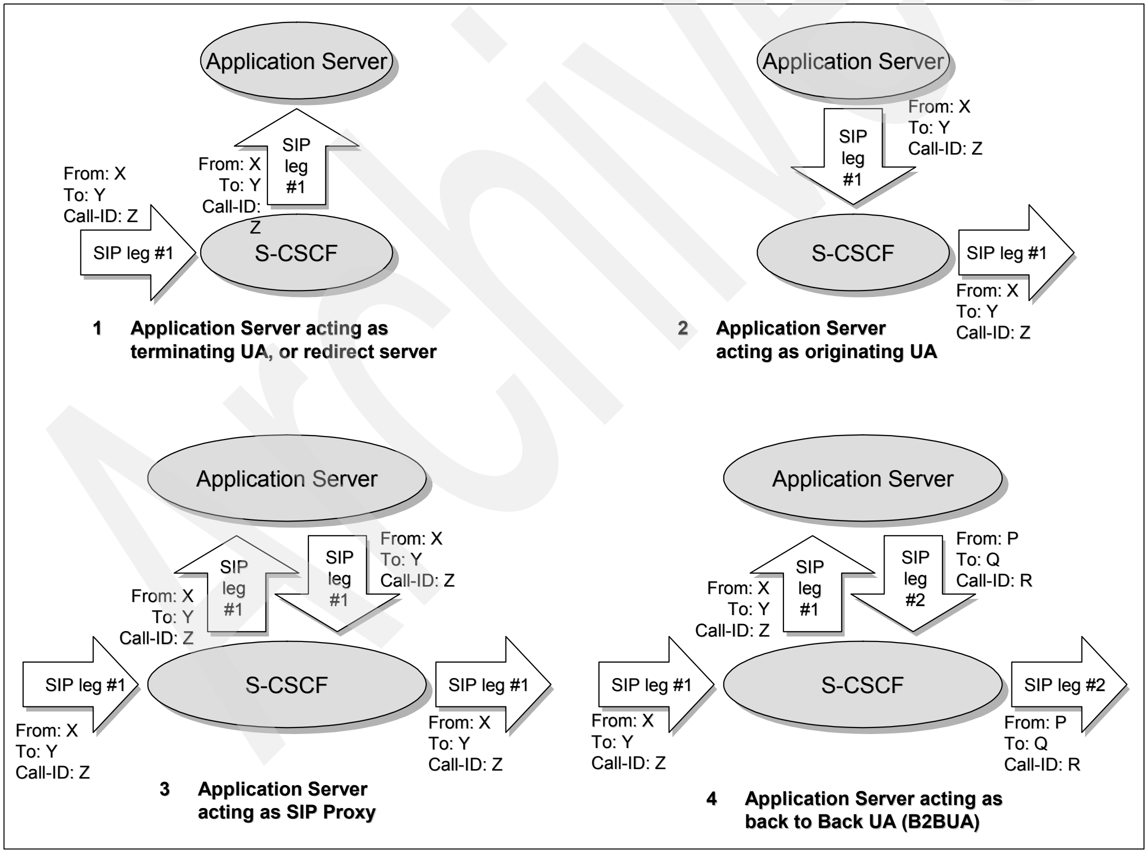


Figure 8-8 Application Server acting as user agent, SIP proxy, or B2BUA

Presentation of SIP Parameters

The ISC interface supports the Service Point Triggers (SPT) for SIP methods such as REGISTER, INVITE, SUBSCRIBE, and MESSAGE. Data in the SPTs include:

- Presence or absence of any header
- Content of any header
- Direction of the request
- Session description information (SDP)
- Priority

An Initial Filter Criteria defines the logical expression for manipulating SIP message method and headers, thereby defining which service platform or platforms are used, and in what order, based on information received by the S-CSCF. Each Initial Filter Criteria is associated to a SIP URI (Application Server URI) to which the request should be forwarded if a match occurs. So when the CSCF receives a request, the iFCs are first evaluated (there can be several, each has a different priority), then the request is forwarded to the appropriate IMS Application Server, such as WebSphere Application Server. The Application Server in the IMS service plane receives the request, applies the business logic for the application, and appropriately routes the request.

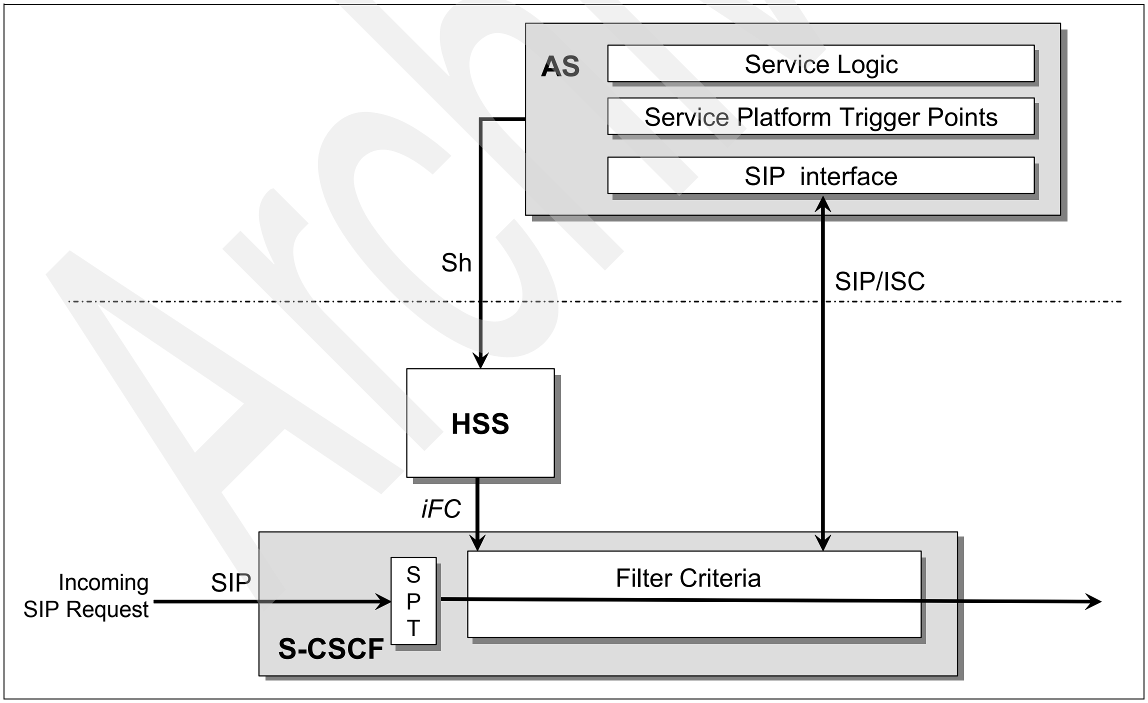


Figure 8-9 ISC Interface

Based on the ISC interface, there are several ways in which the application server is expected to interact with the CSCF, they include:

- Acting as a terminating user agent (UA)
- Acting as an originating user agent to originate traffic on behalf of a user
- Receiving requests
- Serving as a proxy function
- Acting as a third party call control application

3GPP defined a list of Private Headers (P-Headers) to allow for control mechanisms, charging mechanisms, etc. As SIP requests/messages are processed in the IMS control plane, these P-Headers are inserted and are made available to the application server. The application server can then act on them, enhance them and also provide information also using P-Headers.

The following P-Headers are visible only to the application server:

- P-Asserted-Identity (RFC3325)

  Carries valid and authenticated public user identity from the IMS control plane to the application server.

- P-Charging-Vector (RFC3455)

  Carries charging correlation information from IMS control plane to the application server.

- P-Charging-Function-Addresses (RFC3455)

  Carries offline and online charging function addresses from the IMS control plane to the application server.

The following P-Headers are visible to both the application server and the user environment.

- P-Access-Network-Info (RFC3455)

  Carries information of the access network from user environment to the IMS control plane and from the IMS control plane to the application servers (visible only to trusted application servers). Allows the user environment to provide information related to the access network it is using (such as cell ID).

- P-Called-Party-ID (RFC3455)

  Carries the target public user identity from the IMS control plane to the user environment. Allows the terminating user environment to learn dialed public user identity that triggered the call. This field may be seen at the application server when the application server is the called party (such as the destination of the session), but not in other scenarios (such as when the application server is just a proxy in the chain of proxies in the path towards a user environment).

Note: IBM WebSphere IMS Service Control Interfaces Component (also referred to as ISC Interfaces) is an integral part of WebSphere Application Server Version 6.1. It is only licensed for use through the IBM WebSphere IP Multimedia Subsystem Connector Version 6.1.0.

8.3.2 Diameter services

Diameter is an Authentication, Authorization and Accounting (AAA) protocol developed by the IETF. Diameter was initially developed as a second generation protocol to replace the RADIUS protocol, it has since evolved into both a AAA protocol and a more general purpose protocol used to access database information.

In the IMS architecture, Diameter has 3 different types of network nodes:

- Clients

  Clients are the nodes that originate AAA requests.
- Servers

  Servers handle those requests for a particular domain or realm.
- Agents

  Provide relay, proxy or translation functions.

Diameter is a peer-to-peer protocol. In a typical Diameter transaction as illustrated in Figure 8-10, the Diameter client sends requests to the Diameter server, the server then queries the database and sends the response back to the client. The server can also initiate transactions by sending unsolicited messages to the client.

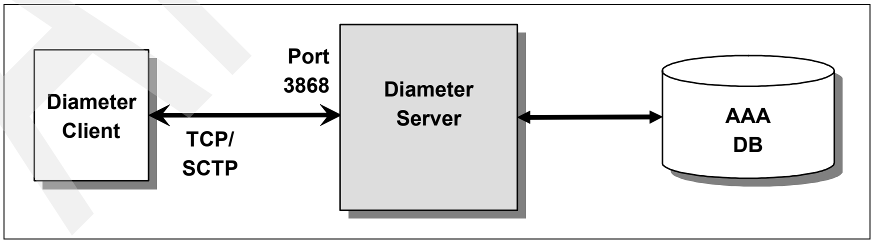


Figure 8-10 Diameter components

Diameter runs on connection oriented transports such as TCP and SCTP, and security is provided through TLS (Transport Layer Security) or IPSec (Network Layer security).

The Diameter protocol is highly scalable. Proxy, relay and redirect agents are used to fan-out the network. Diameter clients are front ended by proxy agents, or relay agents that dynamically relay Diameter requests to appropriate Diameter Servers.

The Diameter protocol can limit the resources in the network by limiting the number of connections between any two peers to a single connection.

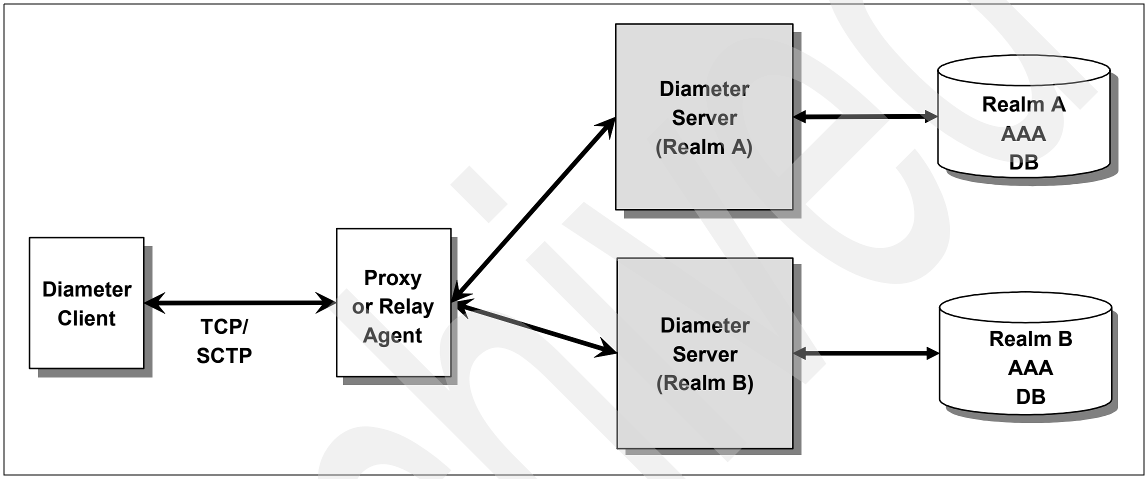


Figure 8-11 Diameter components for scalability/reliability purpose

Diameter protocol is used to implement the following reference points between an Application Server in the service plane and IMS components:

- Sh, and Dh

  Are used between an application server and the HSS and/or the SLF, for user data handling and subscription/notification,

- Rf

  Is used for handling offline charging information to the CCF. Rf supports two offline charging methods that use reply/request pairs:

  – Session Charging

    Configurable charging timer functions:

    • Start - Starts an accounting session timer

    • Interim - Resets the accounting session timer

- Stop - Stops the accounting timer and expires the session
  - Event Charging

    Accounting process in a single operation

- Ro

  Is used for handling online charging

The Diameter Protocol can be used in two different service areas:

- Accounting Services

  This provides the capability to send Accounting-Start, Accounting-Stop, and Accounting-Interim messages as well as an Accounting Event notification to the Accounting server. The Accounting-Start, Accounting-Interim, and Accounting-Stop messages contain the necessary attributes to create Call Data Records (CDRs) to document the services used by a subscriber. Moreover on line charging allows to perform credit control before usage of IMS resources.

- Subscriber Profile Services (IMS-Sh)

  This service provides data contained in XML documents to handle the following functions:

  - Sh-Pull

    To query information pertaining to a specific Subscriber or user

  - Sh-Update

    To update the information retained on the HSS for a specific Subscriber or user

  - Sh-Subs-Notif

    To register for notifications when particular information is updated

  - Sh-Notif

    To inform the Application Server that the information specified by an earlier Sh-Subs-Notif request has been updated

Note: A Web Services interface can be used to provide external access to the Diameter services. This of course requires that the transactions are secure and users authenticated.

8.3.3 IBM WebSphere Diameter Enabler

IBM WebSphere IP Multimedia Subsystem Connector includes the IBM WebSphere Diameter Enabler Component (Diameter Enabler). The Diameter enabler includes Web Services and a client which enable Diameter applications to send and receive subscriber profile information and send accounting information from the Home Subscriber Server (HSS) and to the Charging Collection Function (CCF).

WebSphere Diameter Enabler includes the following:

- WebSphere Diameter Enabler base

  The WebSphere Diameter Enabler acts as a Diameter gateway, receiving and replying to Web service requests on the one side while sending and receiving Diameter packets on the other. This eliminates the need for the IMS Application Server application developer to use the Diameter protocol.

- Rf accounting Web Services

  The Rf accounting Web Services provides an offline charging interface.

  Note: Only IMS-Rf is currently supported, Ro is planned for future release.

- Sh subscriber profile Web Services

  The Sh subscriber profile Web Services provide access to subscriber profile information.

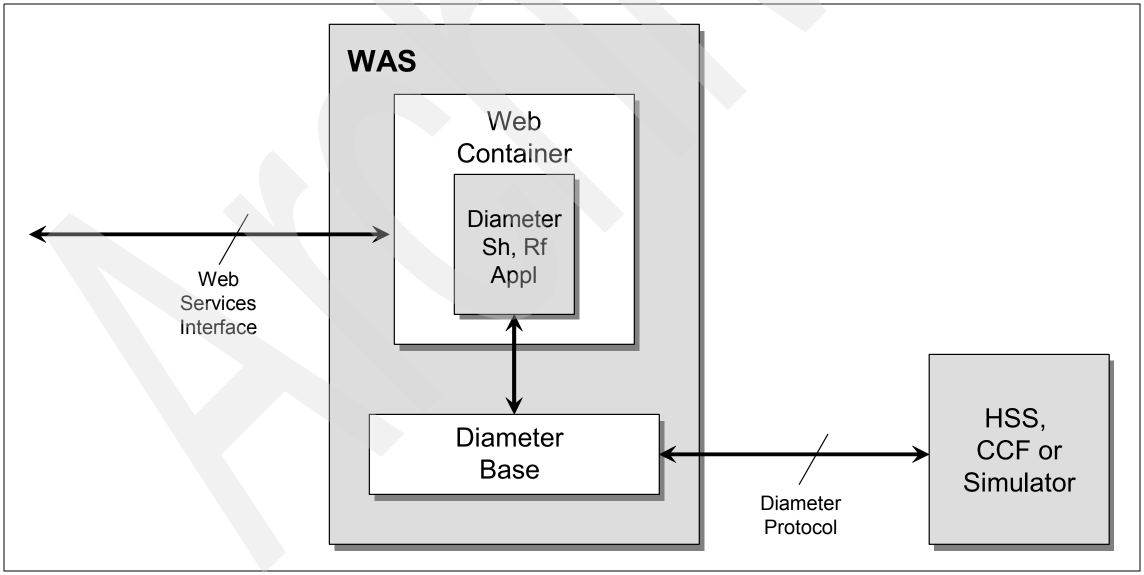


Figure 8-12 Diameter Enabler Component

WebSphere Diameter Enabler provides application programming interfaces (API) using Web Services to help developers rapidly develop and deploy applications that access data from the Home Subscriber Server (HSS) and update data for the Charging Collection Function.

The Web Services Description Language (WSDL) is included for Rf accounting Web Services and Sh subscriber profile Web Services.

Each WSDL file describes the operations, parameters, and data types that comprise the interfaces of the Rf accounting Web Services and Sh subscriber profile Web Services that can be executed by other applications.

8.4 WebSphere Presence Server

The design of an IMS application is facilitated by the use of common service enablers. Service enablers are key elements of IMS architecture they represent generic and reusable building blocks. Two enablers that are essential for IMS solutions are the Presence and Group List Management service enablers.

The IBM WebSphere Presence Server includes IBM WebSphere Presence Server Component (WebSphere Presence Server) the service enabler for Presence, and the IBM WebSphere Group List Server Component (WebSphere Group List Server) the service enabler for Group List Management.

8.4.1 IBM WebSphere Presence Server Component

WebSphere Presence Server adheres to industry standards such as Session Initiation Protocol (SIP) defined by IETF (Internet Engineering Task Force), and the SIP extension SIMPLE (SIP Instant Messaging and Presence Leveraging Extensions). WebSphere Presence Server processes information in PIDF (Presence Information Data Format-RFC3863) and RPID (Rich Presence Information Data Format-draft-ietf-simple-rpid) formats.

It deploys on the IBM WebSphere Application Server platform and utilizes the SIP container in WebSphere Application Server. The WebSphere Presence Server uses JDBC™ data access methods for easy integration with JDBC compliant databases such as IBM DB2 Universal Database and Oracle® Database.

WebSphere Presence Server includes the following IETF defined components:

- Presence server

  The presence server acts as an agent or as a proxy. When acting as a presence agent, it is aware of the presence information of *presentities*. When it acts as a proxy, it sends SUBSCRIBE requests to other entities that acts as presence agents. The presence server component also acts as the presence agent for all SUBSCRIBE requests.

- Event state compositor

  Is a User Agent Server (UAS) that processes PUBLISH requests from presentities, and is responsible for compositing event state into a complete, composite event state of a resource.

The Presence Server is made up of a collection of inter-working services and utilities that provide a number of functions including the following:

- Publish-Subscribe-Notify service

  The Publish-Subscribe-Notify (PSN) Server is responsible for collecting, managing and distributing presence information. It implements the major functionalities of the WebSphere Presence Server in accordance to the following IETF standards, RFC3265, RFC3856, RFC3857, RFC3858, RFC3903, and others still in the draft phase.

- Presence data storage service

  The WebSphere Presence Server allows a presentity to publish a presence document. The WebSphere Presence Server allow presentities to publish presence documents. It stores the data until the document expires or deleted by the presentity.

- Events service

  The WebSphere Presence Server supports subscriptions for notifications to changes in presence information. It receives SIP SUBSCRIBE requests, verifies their compliance to SIP standards (presence of all MUST headers and reasonable syntax of the request), and then creates a subscription on the presence. The WebSphere Presence Server collects the presence documents and sends NOTIFY messages to subscribers.

- Resource List service

  The WebSphere Presence Server provides a service (Resource List Server) that accepts subscriptions to resource lists and notifies the subscribers of the presence state of all entities in the list. This occurs whether the presentities are registered within or outside the resource list domain.

- Location (Contacts) service

  The Location (Contact) Service is an abstract service that generates the registration binding (associates the Address Of Record (AOR) with one or more presence contacts) and stores the registration information in the database. The WebSphere Presence Server associates an AOR with one or more contact addresses received in the REGISTER method. This service may be used by a SIP redirect or proxy server to obtain information about one or more presentity's locations.

Note: The WebSphere Presence Server is extensible, you can extend it by creating the following:

- Services and utilities
- Types of presence information
- Sources of presence information

8.4.2 The Presence Management enabler

The Presence Management enabler (PME) is an application that collects, manages, and distributes real-time presence information to applications and users on an as-needed basis. Presence Management enabler allows applications to both publish presence information and to subscribe for notifications to changes in presence information. When an application subscribes to presence information, the Presence Management enabler will send notifications to the application with the presence information.

The following actors are involved when using a Presence Server:

- Presentity

  A Presence Entity is a resource that provides presence information to a presence server.

- Watcher

  An entity that requests presence information about resources (presentities). It can be either:

  - A subscriber

    Subscribes to presence information of a resource and indicates interest to be notified of subsequent status changes.

  - A fetcher

    Gets a one time notification of a requested resource information (subscription with expire time zero).

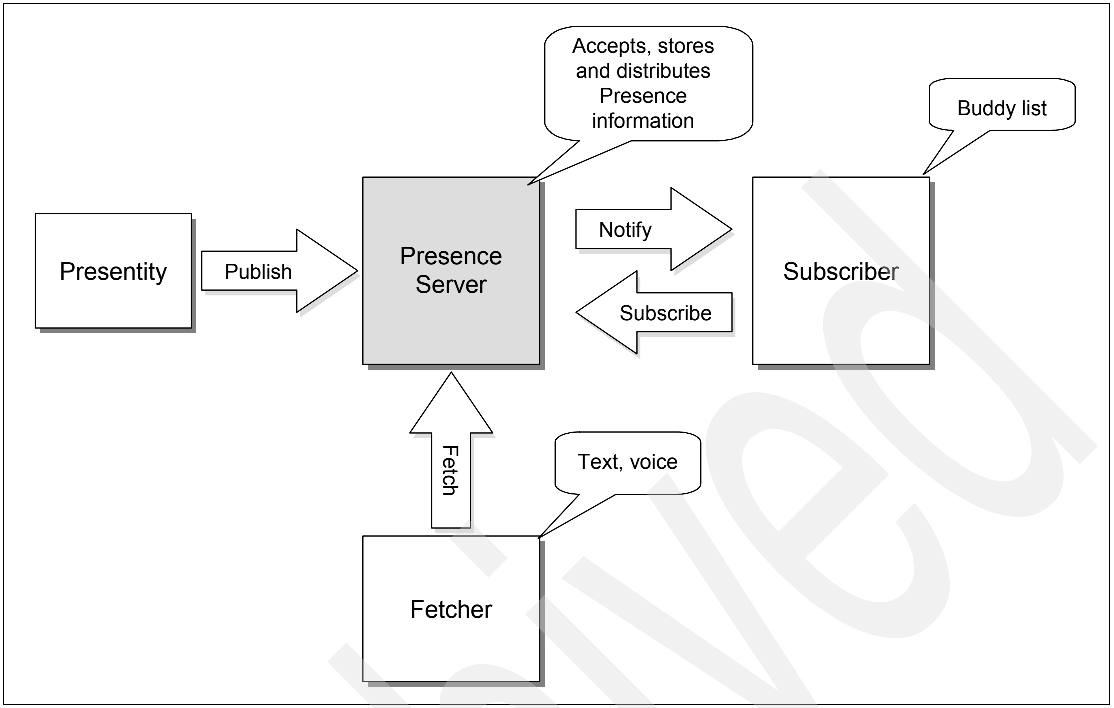


Figure 8-13 Overview of Presence Management enabler

8.5 IBM WebSphere Group List Server

The IBM WebSphere Group List Server (GLS) supports creation of network-based groups, management of the groups, and the administration of membership, privileges and attributes. Members of a group can be either a public identity of a user or a URI that identifies a nested group. Group lists are maintained at the network level, as a result, it can be used from different devices. And because group lists are maintained separately from services, multiple services can use the same group list.

The IBM GLS is a carrier-grade server that leverages scalability and high availability features of IBM middleware product offerings including the IBM WebSphere Application Server Network Deployment V6.1, the IBM Tivoli® Directory Server Version 6.0 for storage of group list information and DB2 Universal Database (DB2 UDB) for persistence of Usage Records and system configuration parameters.

The Group List Server performs a number of functions including the following:

- Provides the common repository for group definitions and lists.
- Makes lists available to devices, applications and services.
- Manages the authorization and permissions for accessing group lists.
- Enables updating of lists through standards-based interfaces.
- Provides subscription capabilities to alert applications and services of changes to lists.

The IBM GLS consists of multiple components and supporting tools as illustrated in Figure 8-14.

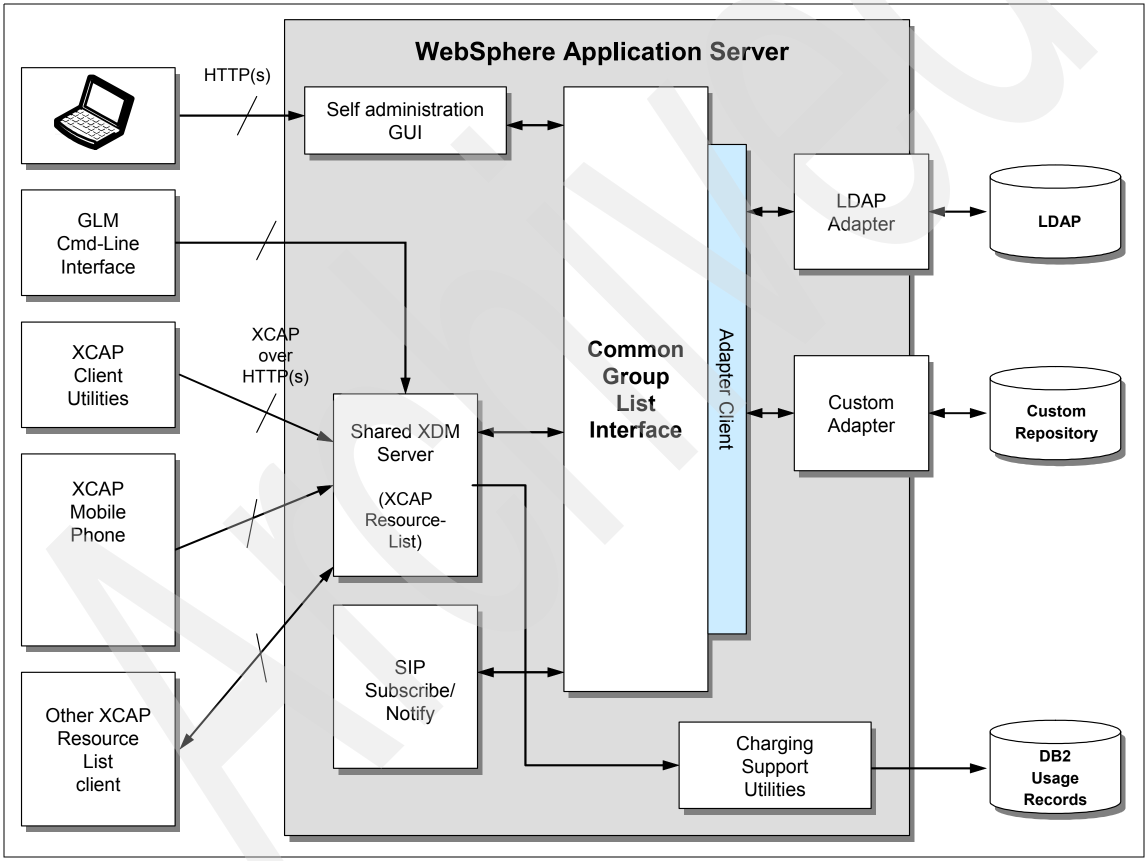


Figure 8-14 IBM GLS Architecture

The implementation of the IBM WebSphere Group List Server Component supports command line and programatic interfaces:

- Command line interface

  Using the IBM WebSphere Group List Server Component command line interface, you can bulk load members, (such as the members of a company directory), delete members, and get membership information about a group.

- SIP interface

  The SIP interface implements the group list subscribe and notify capability. It provide subscribers notification when a group definition is modified in an XCAP document that they own or that they have sufficient rights to view or modify.

- XCAP interface

  The Extensible Markup Language (XML) Configuration Access Protocol (XCAP) server based command line interface provides the following capabilities:

  – Allows access to IBM WebSphere Group List Server Component by mobile device or other applications which support an XCAP client
  – Provides a Ut reference point interface into IBM WebSphere Group List Server Component
  – Can be used by administrators to create and maintain groups in a bulk-load fashion

  IBM WebSphere Group List Server Component also provides a set of XCAP Client utility classes that enable easy development of XCAP client applications, which communicate with the Group List Server through the XCAP interface. Charging support utility classes enable the generation and storage of usage records, which you can use to generate customer charging records. The bulk load scripts use the XCAP Client utility classes to send XML documents that describe the required group management operation through the XCAP interface.

8.5.1 The role of Group List Management

The Group List Management Service Enabler allow users to create groups that can be used for different services. The lists are applicable where public identities are required, examples include instant messaging buddy lists, public or private chat groups, buddy lists used to establish Push-to-talk over Cellular (PoC) sessions.

Groups have the following characteristics:

- Identifier

  Each group has a globally unique, addressable group identifier that is used when it is necessary to refer to a specific group, for example sending a message to a group or subscribing to a group's availability.

- Group information

  Informative text that describe the type and usage of the group.

- Properties

  Used to specify group visibility (who is authorized to see the group), and group duration (expiration time, or by administrator)

- Service specific group information

  Provides additional information about how the group should be used in the context of a specific service.

A Group List Management enabler support secure use and access to group content and notification of changes. It authenticates and authorizes users and applications requesting access to group content. Notifications of group content changes is provided only to authenticated and authorized users and applications.

It also provide offline usage logging to support various charging models such as pay per transaction, volume based charging and indirect charging.

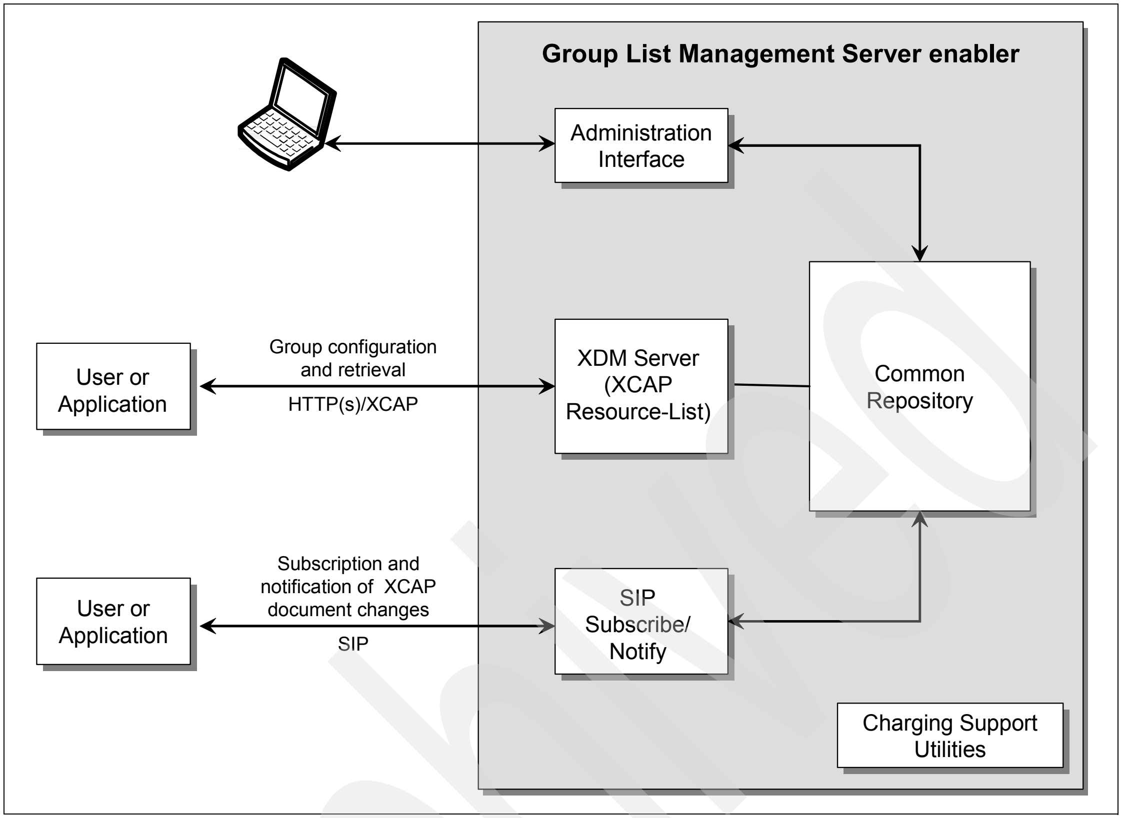


Figure 8-15 Group List Management Server enabler

> **Note:** The Presence Management Enabler (PME) can take advantage of the functions provided by the Group List Management Service Enabler (GLMSE). For example the PME can make use of SIP events to subscribe and get notifications when a subscribed document is changed in the GLMSE. And PME can use XCAP protocol to retrieve documents stored in the GLMSE.
>
> Draft-ietf-simple-event-list-07.txt describes a means for SIP users to send a single subscribe request on a resource that represents a list of resources (resource-list). The PME can utilize the interfaces to the GMSE to receive the resource-list members, including updates on its members, and send notifications including the presence information of all the group members.

8.5.2 XDM/XCAP Interface

The XML Document Management XDM Specification (XDM_Spec) defines the use of the common protocol XML Configuration Access Protocol (XCAP) by which principals can store and manipulate their service-related data as XML documents. It also defines how XCAP URI can be used to identify entire XML documents, individual elements, or XML attributes that can be retrieved, updated, or deleted.

The XML documents are stored on an XCAP server. Each XCAP resource on the XCAP server has an associated application. If any resource is changed, the XCAP server must be able to validate the content of each XCAP document that is being modified. Further, the resource interdependencies and how changes to one resource will effect other resources, has to be determined.

The application must provide the following information in order to use the XCAP resources:

- An Application Unique ID (AUID), which uniquely identifies the application usage
- An XML schema
- The XML document with a well defined semantic
- Default namespace binding, which maps the namespace prefixes to the namespace URIs
- The MIME type of the document
- Naming conventions for XCAP client URIs

> **Note:** Detailed information about the XCAP specification can be found in the IETF draft, *Extensible Markup Language (XML) Formats for Representing Resource Lists*, at:
>
> `http://www.ietf.org/internet-drafts/draft-ietf-simple-xcap-list-usage-05.txt`

The XCAP URI

The XCAP URI (or URL) uniquely identifies XML documents, the type of XML content and optionally, an XPATH which can identify a specific XML node which can be stored or retrieved.

The general form of an XCAP URI is as follows:

<XCAP Root>/<AUID>/<Document Selector>/~~/<Node Selector>

Where:

- <XCAP Root>

  Identifies the HTTP request URL and context root.

  Example: `http://myhost.ibm.com:9080/services/`

- <AUID>

  Is the XCAP Application User IDentifier. This identifies the type of XML document.

  Examples: `resource-lists or rls-services`.

- <Document Selector>

  Is the portion of the URI which identifies the specific document to be stored or accessed. Documents are segregated into two types:

  – "users"

    The "users" type contains user documents which are created and maintained by specific users. The form of the *<Document Selector>* for documents contained in the users branch is "*users/<XUI>/<Document Name>*", where *<XUI>* is the XCAP User Identifier, and *<Document Name>* is the name of the document.

  – "global"

    The “global” type contains documents that ate global to the Group List Management enabler. For the global branch the *<Document Selector>* for documents contained in this branch is "*global/<Document Name>*", where *<Document Name>* is the name of the document.

- <Node Selector>

  Is an expression which can be used to identify a specific XLM element or attribute which is to be updated or retrieved.

The following are two examples of the XCAP URI:

- User-specific Document
  `URL:http://myhost.ibm.com:9080/services/resource-lists/users/sip:joe@foo.com/MyGroup.xml`

- Global document

  A node URL pointing to a list element with name “Accounting” within a global document:
  `http://myhost.ibm.com:9080/services/resource-lists/global/WorkContacts.xml/~~/resource-lists/list[@name=“Accounting”]`

XCAP Usage

Group List Server uses the xcap-caps usage to retrieve the AUIDs that the XCAP server implementation supports. Group List Server retrieves the XCAP capabilities document by using the following XCAP URI:

- Resource-lists

  Group List Server uses the resource-lists usage as its primary method of managing group lists.

- RLS services

  Group List Server uses the rls-services usage to map a user-defined SIP URI that identifies a resource list. XCAP URI are used to access the resource list. You can store individual rls-services documents using the following XCAP Web address:

  `xcap_root/rls-services/users/XUI/index`

  Where:

  – xcap_root

    Is the root of the tree within the domain where all the XCAP documents are stored

  – XUI

    Is the XCAP User Identifier

- org.openmobilealliance.xcap-directory

  Group List Server uses the org.openmobilealliance.xcap-directory usage to retrieve an XDM document directory XML file. Group List Server generates the XDM document directory when it receives a GET request for the following XCAP URI:

  `xcap_root/org.openmobilealliance.xcap-directory/users/XUI/directory.xml`

  Where:

  – xcap_root

    Is the xcap_root_URL that specifies the `hostname:port/context_root`

  – XUI

    Is the XCAP User Identifier

- XCAP-CAPS

  Group List Server uses the xcap-caps usage to retrieve the AUIDs that the XCAP server implementation supports. Group List Server retrieves the XCAP capabilities document by using the following XCAP URI:

  `xcap_root/xcap-caps/global/index`

Where:

– xcap_root

  Is the `xcap_root_URL` that specifies the hostname:port/context_root of the XCAP Server

– XUI

  Is the XCAP User Identifier

8.6 Telecom Web Services Server

Telecom Web Services Server (TWSS) utilizes service-oriented architecture to provide a platform whereby Service Providers can provide third parties a secure, reliable, and policy driven access to telecommunication network capabilities, such as messaging, location, presence, and call handling.

The TWSS physical architecture is illustrated in Figure 8-16 on page 194 it consists of the following:

- Telecom Web Services Access Gateway which deploys on a WebSphere ESB Version 6.0.2.7 and WebSphere Application Server Version 6.0.2.7
- Telecom Web Services implementations deploy on a WebSphere Application Server 6.1
- Service Policy Manager deploys on a WebSphere Application Server 6.1. It can be installed on the same physical server as the Telecom Web Services implementations.

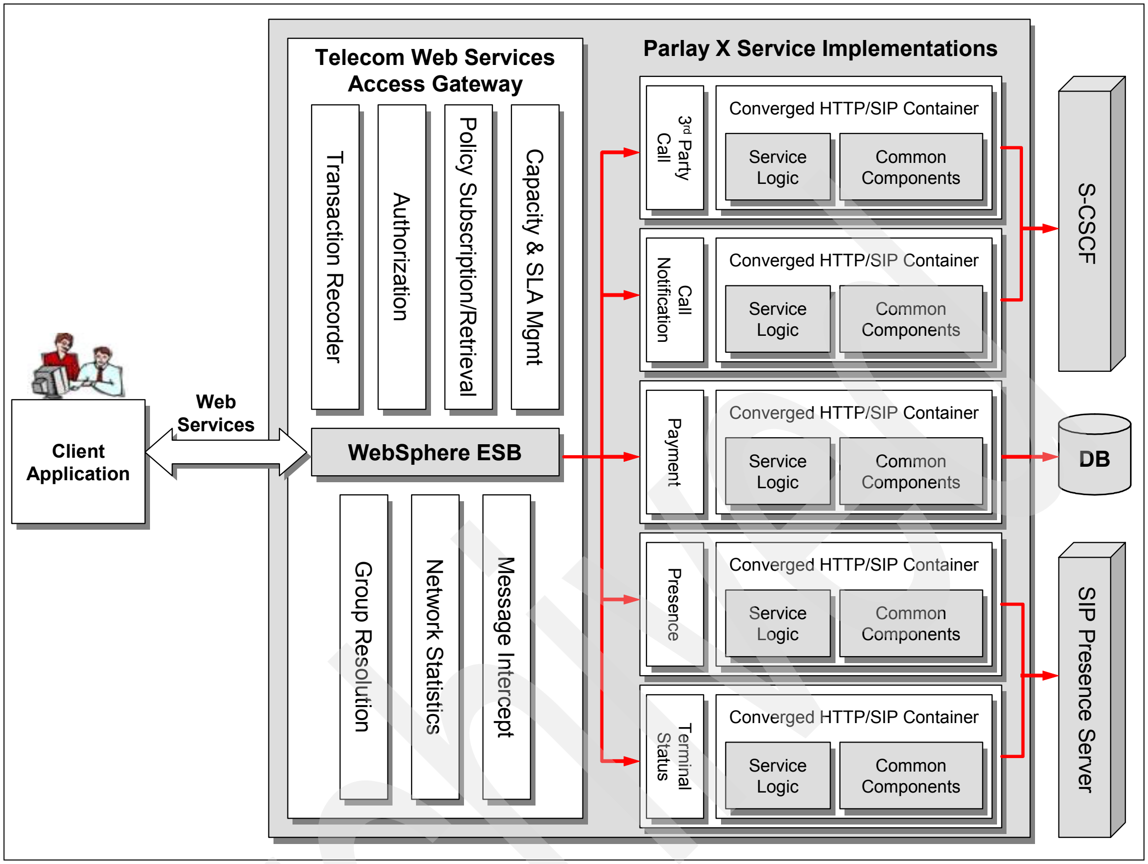


Figure 8-16 WebSphere Telecom Web Services Server architecture

8.6.1 Telecom Web Services Access Gateway

The Access Gateway is the control point for network access. It is built on WebSphere Enterprise Service Bus (ESB) which provides the flexibility for constructing tailored Web Services message processing in accordance with the service provider's network policies. It provides pluggable ESB components and default message flows.

The Web Services message processing and flow can be modified in WebSphere Integration Developer as Mediation flows, with the use of graphical programming techniques. You can insert additional mediation primitives anywhere in the flow, change the order of execution and so on.

A overview of the architecture for the Telecom Web Services Access Gateway is illustrated in Figure 8-17 on page 195.

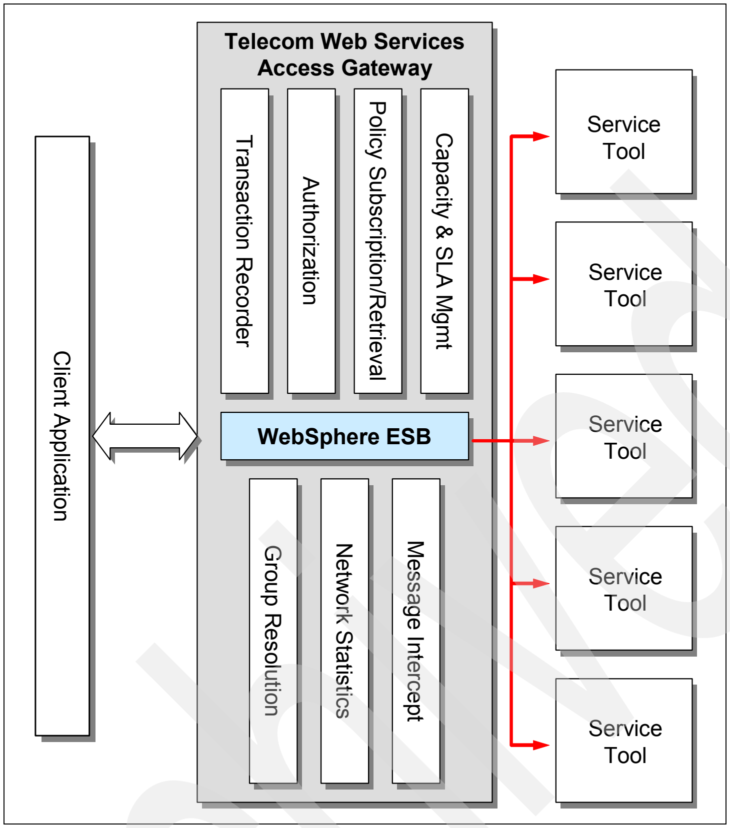


Figure 8-17 The Telecom Web Service Access Gateway architecture

Telecom Web Services Access Gateway provides a default flow implementation. The default flow is illustrated in Figure 8-18 on page 196. The following components are considered mandatory for a base Telecom Web Services Access Gateway configuration and flow:

- Transaction recorder mediation primitive
- Policy/Subscription mediation primitive
- Service invocation mediation primitive

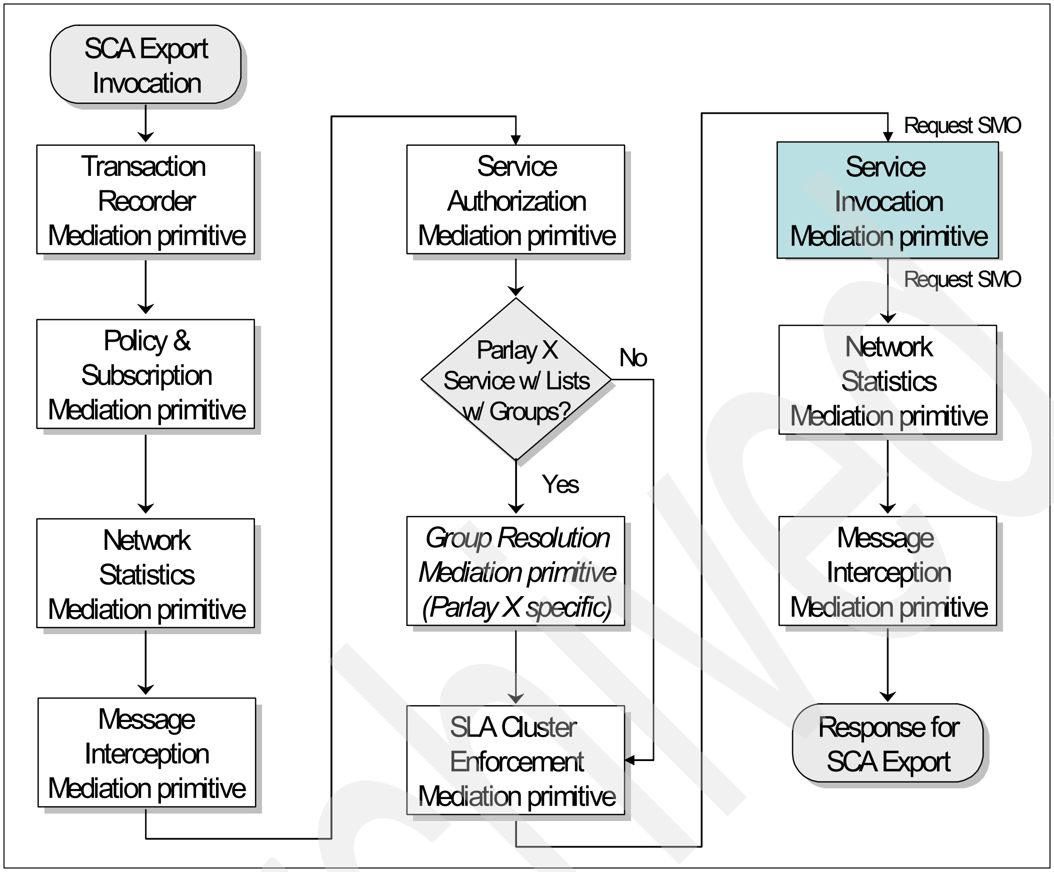


Figure 8-18 TWSS Access Gateway default mediation flow logic

8.7 Telecom Web Services Server service implementations

The Telecom Web Services Server supports service implementations that are accessible through Web Services. The services range from SIP IMS enabled services to SIP Parlay based services and direct connect protocol services.

SIP based services can directly access IMS components through WebSphere Application Server IMS support. Parlay based services need a Parlay Connector to communicate with a Parlay Gateway to access core network function. Direct connect services need specific connectivity programming with various telecommunication-specific protocols.

Note: Parlay X implementations utilize OSA/Parlay gateways for connectivity to telecommunication networks. Direct connection to data services network elements through standard IP protocols such as SMSC via SMPP are planned for future releases.

To facilitate the rapid development of new service implementations, Telecom Web Services Server service implementations include several reusable components which provide common services intended to be shared by all service implementations.

Web Services Implementation

The following Parlay X 2.1 are implemented out of the box with TWSS

- Third Party Call

  It uses S-CSCF as a SIP proxy and enables third party client applications to initiate a call from a network entity between two completely different users or user agents

- Call notification

  It uses S-CSCF as a SIP proxy and enables third party client applications to get call event information from the network, either through looking up the information or by asking for notifications. The type of events that can be reported include:

  – The called party is on the phone
  – The called party does not answer the phone
  – The called party is unreachable
  – A call was attempted

- Payment

  It enables third party client applications to send payment information to the service provider that is determined by the service configuration information, by enabling the application to charge an amount against a user account or place money into a user account. The Web service then writes this charging information to a local database, from which billing records are later generated offline.

- Presence

  It enables third party client applications to get information from the presence server, either by looking up information or asking for notifications. Presence service implementation allows client applications to use Web Services to subscribe to a presentity, synchronously query the current presence information for a presentity and to unsubscribe from a presentity.

- Terminal Status

  Allows client applications to use Web Services with a presence server to request the status of an IMS terminal (or terminals) and receive notification for changes to the state of the terminal (or terminals). This service also supports group-level operations.

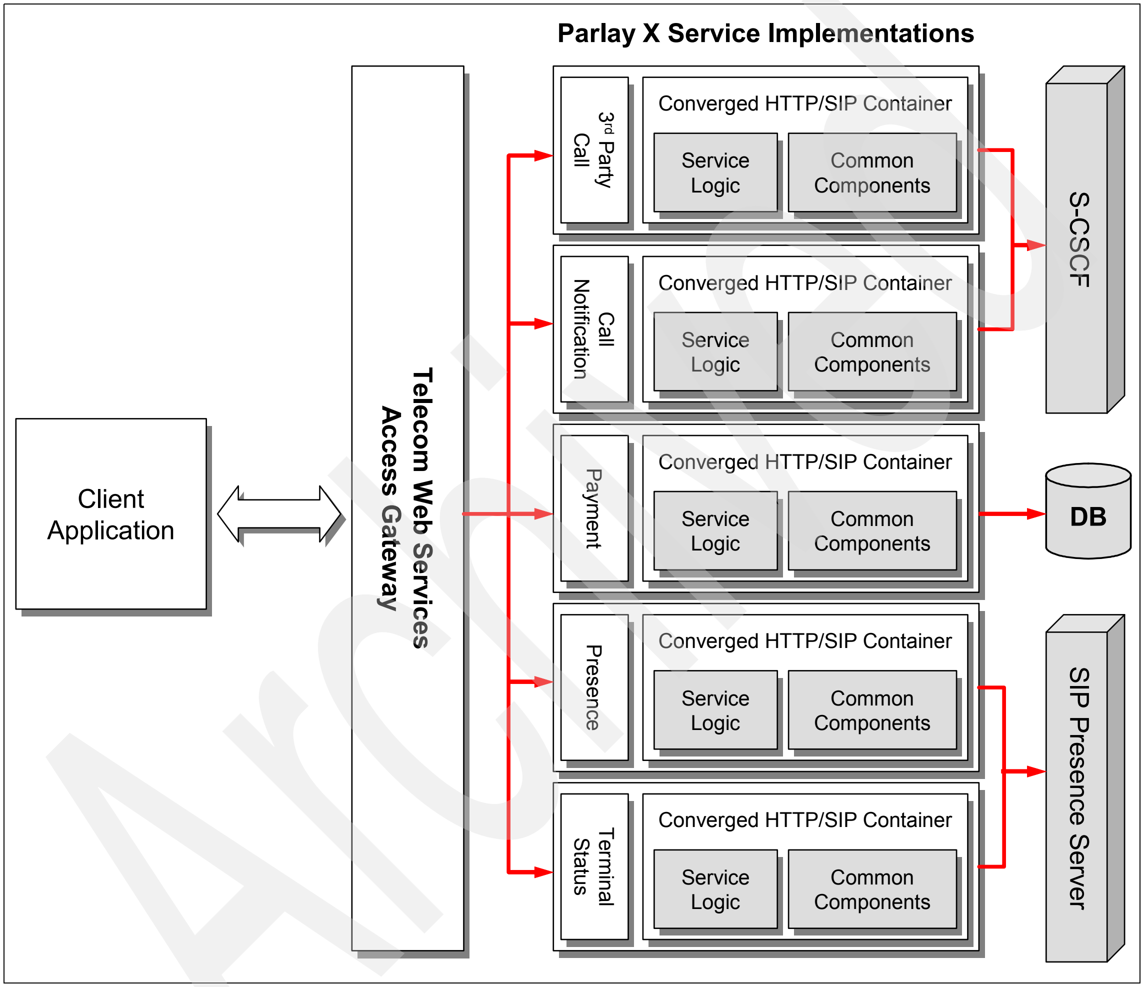


Figure 8-19 Back-end Service Implementations

8.7.1 Common components

The Telecom Web Services service implementations includes reusable components that provide functionality in the following areas and facilitate rapid development:

- Admission Control

  It controls incoming admission of Web Service traffic based on TWSS platform capacity so that it does not exceed a configured rate for a given time interval.

- Traffic Shaping

  It controls the rate at which traffic can be directed towards an element in the network. Each network element can be associated with a resource definition defining the rate of traffic that can be directed towards the network element. Traffic shaping will employ a token bucket-based mechanism to allow for limiting traffic burst size and average rate.

- Parlay X Notification Delivery

  It provides facilities for the delivery of notifications to the destination endpoint through the front-end TWS B2B gateway.

- Faults and Alarms

  When encountering error conditions, service implementations output fault information and potentially emit an alarm for severe error conditions. Fault and alarm information will both be emitted in Tivoli's common base event (CBE) format supported by WebSphere Application Server and via JMX management notifications.

- Service Usage Record

  Each service implementation generates a service usage record that describes how the service was used for accounting and billing purposes. Service usage records are stored in relational table format such that service usage records can be searched via SQL queries.

8.7.2 Service Policy Manager

Service Policy Manager provides administration interfaces that you can use. It also includes storage capability and access mechanism to enable the definition of requesters, services, as well as subscriptions that associate services with requesters. It utilizes a Web interface and administrative Web Services to enable interaction with the Service Policy Manager data store.

8.8 WebSphere Enterprise Service Bus

An Enterprise Service Bus ESB supports the concepts of SOA implementation by:

- Decoupling the consumer's view of a service from the actual implementation of the service

- Decoupling technical aspects of service interactions
- Integrating and managing services

These concepts are achieved by replacing direct connections between service consumers and providers with a hub and spoke architecture.

WebSphere Enterprise Service Bus is designed to provide the core functionality of an ESB for a predominantly Web Services based environment. It is built on open standards. Based on WebSphere Application Server Network Deployment, it inherits the built-in messaging provider and quality of services. WebSphere Enterprise Service Bus adds a mediation layer based on the Service Component Architecture (SCA) programming model on top of this foundation to provide intelligent connectivity.

SCA was developed to simplify the integration between business applications and the development of new services. It defines an implementation of service-oriented architecture (SOA). SCA separates application business logic and the implementation details by providing a model that defines interfaces, implementations, and references in a technology neutral way, enabling the binding of these elements to any technology specific implementation.

Business data that is exchanged in an integrated application in WebSphere Enterprise Service Bus is represented as business objects. Business objects are based on Service Data Objects (SDOs), which is used to describe complex data structures and provide a universal means for representing and accessing data.

WebSphere Enterprise Service Bus uses the Common Event Infrastructure (CEI) to provide event management services, such as event generation, transmission, persistence, and consumption.

WebSphere Enterprise Service Bus supports mediation of interactions between endpoints beyond protocol transcoding. It enables handling of integration logic processing in the ESB instead of in the interacting endpoints. Pre-built mediation functions allow mediations to be visually composed.

The development tool for WebSphere Enterprise Service Bus is the WebSphere Integration Developer introduced in Chapter 6, “IBM WebSphere Integration Developer” on page 119.

For more information about WebSphere ESB V6, see the redbook *Getting Started with WebSphere Enterprise Service Bus V6* at:

http://www.redbooks.ibm.com/abstracts/sg247212.html?Open

8.9 WebSphere Process Server

WebSphere Process Server is built on top of WebSphere Enterprise Service Bus and adds a business process runtime. It offers robust process automation, advanced human workflow, business rules, application to application (A2A), and B2B capabilities — all on a common, integrated SOA platform with native Java Message Service (JMS) support.

The business process component in WebSphere Process Server implements a WS-BPEL-compliant process engine. It represents the fourth release of a business process choreography engine on top of the highly scalable WebSphere Application Server. WS-BPEL defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners.

Human task support expands the reach of WS-BPEL to include activities requiring human interaction as steps in an automated business process.

WebSphere Process Server provides a business state machine component that can be used to model heavily event-driven business process scenarios.

WebSphere Process Server contains a business rule component that provides support for rule sets (If/Then rules) and decision tables.

For more information about WebSphere Process Server V6, see the Redpaper *Technical Overview of WebSphere Process Server and WebSphere Integration Developer* at:

http://www.redbooks.ibm.com/abstracts/redp4041.html

Part 3

SIP applications

Part 3 provides the programming guide for developing SIP based converged applications. It includes working examples that demonstrate use of IBM tools for application development.

9

Developing SIP applications

This chapter introduces the key elements of the SIP Servlet API Version 1.0 and describes considerations for developing SIP Servlets to be deployed in the WebSphere Application Server V6.1.

This chapter contains the following:

- Overview of SIP applications
- Elements of SIP applications
- Best practices

9.1 Overview of SIP applications

The SIP Servlet API specification defines a SIP Servlet as a Java-based application component which is managed by a SIP Servlet container and performs SIP signalling. In this section, we introduce these components and provide an overview of their functionality.

9.1.1 SIP Servlet container

The SIP Servlet container is responsible for receiving and sending SIP messages over the network. The servlet container chooses the SIP Servlet methods, and the order in which the SIP Servlets will be invoked. The selection is based on the contents of the received SIP messages and the set of policies in effect at the container. The servlet container also authenticates and authorizes requests prior to dispatching the requests to SIP Servlets.

Figure 9-1 shows the relationship between a SIP client, the SIP container and the SIP Servlet.

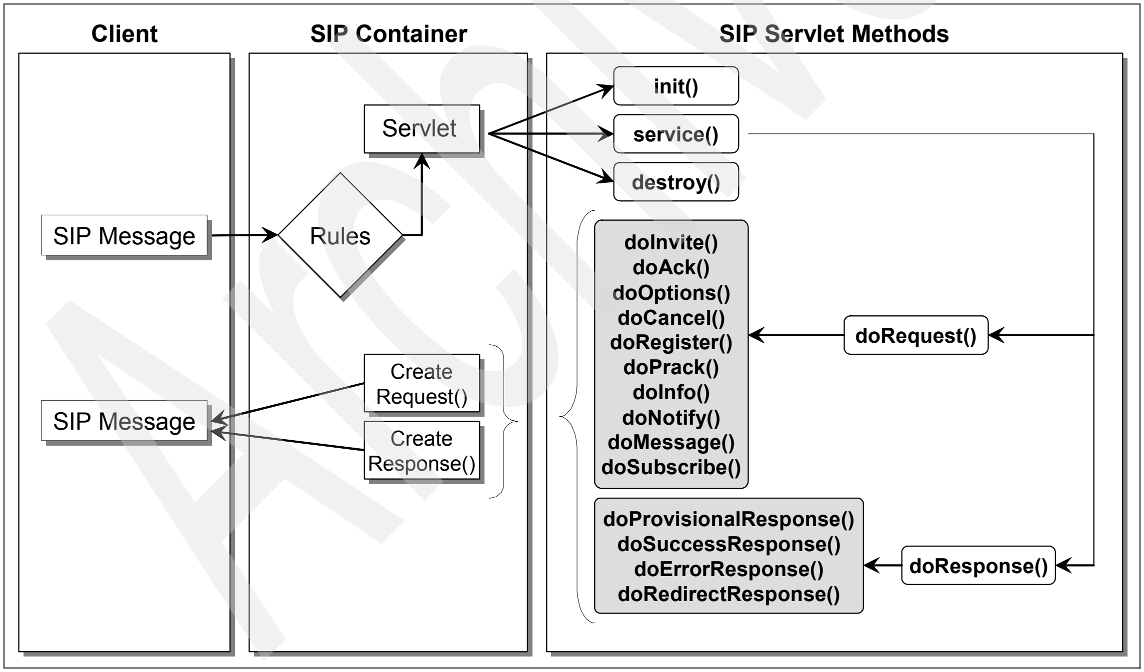


Figure 9-1 SIP Servlet overview

9.2 SIP Servlet

SIP Servlets are grouped together into applications which are deployed to SIP Servlet containers using deployment descriptors.

SIP Servlets respond to SIP events, both requests and responses that are dispatched by the SIP Servlet container. It uses a high level API to create new SIP messages and to inspect received messages. The servlet container ensures that only valid SIP messages that conform to the SIP protocol are created.

The deployment descriptor contains information about how to invoke servlets and rules for mapping SIP messages. The servlet invocation information is used by the container to invoke SIP Servlets within a SIP application. The mapping rules specify how the servlet container should map SIP messages to SIP Servlets and how the security should be enforced.

The SIP Servlet API is built on the generic servlet API, `javax.servlet` package. It adds the `javax.servlet.sip` package which defines SIP specific elements.

9.2.1 Differences between SIP and HTTP Servlet

The SIP and HTTP Servlet programming models share many similarities. However there are a number of important differences:

- Requests and responses

  HTTP Servlets provide a single HTTP response as a result of receiving an HTTP request. An HTTP Servlet overrides the `service (HttpServletRequest request, HttpServletResponse response)` method of the `HttpServlet` class. It is responsible for creating a response using the HttpServletResponse object that is passed in the `service` method invocation.

  In contrast, the receipt of a single SIP request by the SIP Servlet may result in zero or more responses. SIP Servlets decouple the receipt of requests from the generation of responses. They also incorporate the ability to initiate or receive SIP requests and responses. Though a SIP Servlet may override the `service (SipServletRequest request, SipServletResponse response)` method, only one of the objects is populated by the SIP container, depending on whether a SIP request or response is being delivered. The other object is set to `null`. This allows a given request to be decoupled from the response, and for a SIP Servlet to choose how to respond to the receipt of a given message.

- Methods

  The HTTP protocol defines a series of methods such as `GET`, `POST` and `HEAD`. An HTTP Servlet may choose to override the implementation of the corresponding methods such as `doGet (HttpServletRequest request, HttpServletResponse response)`, which is invoked when a request corresponding to the protocol method is received.

The SIP protocol defines a different set of methods which include INVITE, CANCEL and BYE, and the SIP Servlet defines the corresponding set of methods that may be overridden to handle these requests.

Because the generation of SIP responses are decoupled from the receipt of requests, a SIP Servlet overrides methods such as `doInvite (SipServletRequest request)` to handle requests for specific protocol methods, and it uses `doSuccessResponse (SipServletResponse response)` to handle responses for particular classes.

- Synchronicity

  HTTP Servlets handle request messages synchronously, generating responses that are sent to the Web browser before exiting the service method. Should a HTTP servlet not create a response, the HTTP Servlet container will create a default response and return it to the HTTP client.

  SIP Servlets are not required to respond to every request. A SIP Servlet may respond immediately to a request, or not at all. It can perform some other operation such as forwarding the request to a proxy that will respond to the request at a later point in time.

- Application composition

  In response to an HTTP request, the HTTP Servlet container selects and invokes just one HTTP Servlet which generates a single HTTP response. If it is necessary to perform additional operations on the request or response, then one or more HTTP Servlet filters are used to operate on the request flow.

  SIP Servlets on the otherhand support invocation of multiple SIP Servlets. Mapping rules specify one or more SIP Servlets that may be invoked for a given request. Rules make use of different aspects of requests and boolean logic to map requests to the appropriate SIP Servlet. This provides significantly more flexibility when compared to HTTP Servlet filters.

- Session management

  A HTTP client session is generally uniquely identified using a session identifier contained in a session cookie or URL. A `HttpServlet` may store server-side state using a `HttpSession` object and refer to this information when a subsequent request from a client is received.

  The SIP protocol defined sessions between User Agents as SIP dialogs. SIP Servlets maintain state information in SIP dialogs through the `SipSession` interface. The SIP Servlet may access previously stored information as it processes requests and responses in a given dialog through the `SipSession` interface.

9.2.2 Converged servlet

The converged servlet is an IBM extension to the functionality provided by the SIP Servlet API. It facilitates the development of converged applications which combine access through multiple interfaces, for example, it enables combination of telephony and Web elements in a single integrated application where an inbound SIP phone call can display the callers information on a Web page.

Converged servlets can receive and respond to HTTP requests as well as initiate SIP requests while sharing context and session information with other SIP Servlets deployed within the same application.

9.3 Elements of SIP applications

A SIP application is comprised of a number of different elements ranging from requests and responses to events and sessions. The key elements are presented here.

9.3.1 Receiving requests

SIP requests received by the container are dispatched to one or more `SipServlets` based on the mapping rules defined in the sip.xml deployment descriptor. These rules are described in 9.3.7, "Mapping requests to servlets" on page 219.

Request handling methods

When a request is received by the container, it is dispatched to a Servlet, and the `SipServlet` method corresponding to the SIP method of the request is invoked. The mapping between `SipServlet` and SIP methods is shown in Table 9-1.

Table 9-1 SipServlet to SIP method mapping

SipServlet method	SIP method
doInvite	INVITE
doAck	ACK
doOptions	OPTIONS
doBye	BYE
doCancel	CANCEL
doRegister	REGISTER

SipServlet method	SIP method
doPrack	PRACK
doSubscribe	SUBSCRIBE
doNotify	NOTIFY
doMessage	MESSAGE
doInfo	INFO

An application that responds to SIP methods other than those listed in Table 9-2 on page 216 must implement the `doRequest` method which is invoked in response to all SIP requests received by the SIP container.

Example 9-1 shows a simple SipServlet which responds to a SIP OPTIONS request.

Example 9-1 Example OptionsSipServlet

```
public class OptionsSipServlet extends SipServlet {

   protected void doOptions(SipServletRequest request) throws
ServletException, IOException {
      SipServletResponse response = request.createResponse(200);
      response.send();
   }

}
```

9.3.2 Parsing messages

`SipServletRequest` and `SipServletResponse` implement the `SipServletMessage` interface, which provides methods to access, parse and modify message headers and contents.

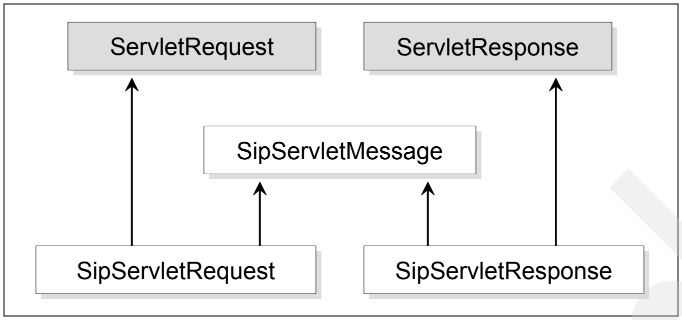


Figure 9-2 Request-response hierarchy

String headers

The contents of a single header can be retrieved as a `String` using the `getHeader(String name)` method. Where a header appears multiple times, this method will return the contents of the first appearance of the header. The contents of all headers can be retrieved using the `getHeaders(String name)` method which returns a `ListIterator` with the contents of each of the headers. The list of headers contained in the message can be obtained by using the `getHeaderNames()` method which returns an `Iterator` containing the header names.

Example 9-2 Parsing SIP headers

```
String expires = req.getHeader("Expires");
ListIterator it = req.getHeaders("Contact");
Iterator it = req.getHeaderNames();
```

Address headers

Several methods are available for obtaining addresses contained in the SIP message headers. The address for a single header can be obtained by using the `getAddressHeader(String name)` method. Similar to the handling of headers as strings, `getAddressHeader()` method returns the address for the first appearance of a header. `getAddressHeaders(String name)` method returns a `ListIterator` of all addresses.

System headers

System headers are those headers that are managed by the container, and are not available for direct manipulated by the application. These headers are the `Call-ID`, `From`, `To`, `CSeq`, `Via`, `Record-Route`, `Route` and `Contact` headers when in the context of a signalling session. For example, these headers can be set when used in the context of a REGISTER request or response, and when responding with a redirection (3xx) or 485 Ambiguous response, but not in other circumstances.

Message content

The `SipServletMessage` interface provides several methods for parsing message content. The MIME type of the content is obtained using the `getContentType()` method. The length of the content is obtained using the `getContentLength()` method.

Contents are obtained by using the `getContent()` method which returns an Object. The type of the object depends on the content type. If the message is of type `text/plain` or any other text MIME type, a `String` is returned.

An application wishing to obtain the raw content without the overhead of parsing, can do so by using the `getRawContent()` method which returns a `byte[]`. This is useful for Back to Back User Agents and applications that want to copy the contents of a message unchanged.

Transport information

The `SipServletMessage` interface provides a series of methods for obtaining information about the transport channel that a message was received on.

The local address and port that a message was received on is available from the `getLocalAddr()` and `getLocalPort()` methods, while the remote address and port that a message was sent from is available from the `getRemoteAddr()` and `getRemotePort()` methods.

The application may determine the transport that was used, for example UDP, TCP or TLS using the `getTransport()` method.

The transport information delivered to the application refers to the transport between the container and the previous SIP server. Hence, if the Stateless SIP Proxy is deployed, an application using this information shall receive the information on the transport between the Stateless SIP Proxy and WebSphere Application Server.

9.3.3 Creating responses

A SIP Servlet can respond to a request that it received by using the `createResponse(int statusCode)` method that is available from the `SipMessage` interface.

`SipServletResponse` provides a set of constants for use as numeric constants for response codes. For example, the constant `SipServletResponse.SC_RINGING` can be used to indicate that the SIP endpoint is ringing in place of specifying the corresponding numeric response code of 180.

Once created, the SipServletResponse can be modified prior to being sent. Headers can be added using the addHeader(String name, String value) and addAddressHeader(String name, String Address) methods. The message content may also be specified using the setContent(Object object, String mimeType) method.

To send the response, the send() method is invoked. The container is responsible for any retransmission of any success (2xx) responses required by the SIP protocol.

9.3.4 Creating requests

To create a SIP Request as a User Agent Client, the client first has to obtains a SipFactory and then create a request.

Obtaining the SipFactory

In order to create an initial SIP request as a User Agent Client, the SIPFactory interface is first obtained from an attribute named javax.servlet.sip.SipFactory from a SIP Servlet's ServletContext. The SipServlet's SIP_FACTORY field also contains a String with same value that may be used to obtain this attribute. Example 9-3 shows code snippet that uses this approach.

Example 9-3 Obtaining a SipFactory from the ServletContext

```
ServletContext context = getServletContext();
SipFactory sipFactory = (SipFactory)context.getAttribute(SIP_FACTORY);
```

Creating Initial Requests

The SipFactory provides several createRequest methods which can be used to create an initial request. The parameters for createRequest are as follows:

- SipApplicationSession

  The SipSession the request will belong to

- method

  The SIP method for the request

- from

  The contents of the From header

- `to`

  The contents of the To header

There are several option for specifying the From and To parameters:

- `SipServletRequest createRequest(SipApplicationSession appSession, String method, Address from, Address to);`
- `SipServletRequest createRequest(SipApplicationSession appSession, String method, URI from, URI to);`
- `SipServletRequest createRequest(SipApplicationSession appSession, String method, String from, String to) throws ServletException;`

Once the `SipServletRequest` is created, it can be modified as required, and then sent using the `send()` method.

Example 9-4 Creating and sending SipServletRequest from a SipFactory

```
URI from = sipFactory.createURI("alice@example.com");
URI to = sipFactory.createURI("bob@example.com");

SipServletRequest invite = createRequest(appSession,"INVITE",from,to);
invite.setRequestURI(to);
invite.send();
```

Selecting the SipServlet to receive the response

When an initial `SipServletRequest` is created, the first `SipServlet` listed in the SIP deployment descriptor is assigned to handle responses. To override this, a handler can be assigned to the `SipSession` for a request. The sample code in Example 9-5 shows how to set the handler for the session with the `servlet-name` of `ThirdPartyCallController`.

Example 9-5 Specifying the handler for responses

```
SipServletRequest invite = createRequest(appSession,"INVITE",from,to);
SipSession session = invite.getSession();

try {
   session.setHandler("ThirdPartyCallController");
} catch (ServletException e) {
   //handle mismatch between deployment descriptor and application
}

invite.setRequestURI(to);
invite.send();
```

Example 9-6 shows how the `<servlet-name> can` refer to a particular `<servlet-class>` in the SIP deployment descriptor.

Example 9-6 Setting <servlet-name> in the SIP deployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sip-app PUBLIC "-//Java Community Process//DTD SIP
Application 1.0//EN" "http://www.jcp.org/dtd/sip-app_1_0.dtd">
<sip-app>
   <display-name>Third Party Call Controller Application</display-name>
   <servlet>
      <servlet-name>ThirdPartyCallController</servlet-name>
      <display-name>Third Party Call Controller</display-name>
      <servlet-class>com.itso.sg247255.CallController</servlet-class>
   </servlet>
...
```

Tip: We recommend that a handler is explicitly set when an initial request is created rather than relying on a particular SipServlet being listed first for an application in the SIP deployment descriptor.

Creating Subsequent Requests

When a `SipServlet` is within a dialog, it may send subsequent requests using the `createRequest` method on the `SipSession` that corresponds to the dialog

```
SipServletRequest createRequest (String method);
```

The container will create a `SipServletRequest` which meets the SIP protocol requirements for subsequent requests within a dialog. The application may modify the request further, it is then sent by invoking the `send()` method.

Sending CANCEL

A User Agent Client may need to cancel an INVITE request that is in progress. The User Agent Client invokes `createCancel()` on the original INVITE `SipServletRequest`. The SIP container is responsible for delaying the transmission of the cancel until a 1xx response has been received.

Back to Back User Agents

In order to ease development for Back to Back User Agents, a `SipServlet` may create a new `SipServletRequest` based on an existing `SipServletRequest`. The method `createRequest(SipServletRequest origRequest, boolean sameCallId)` may be used. This will create a new request which differs from the original request in the following ways:

- The Call-ID is set to a new Call-ID if requested by the `sameCallID` method parameter
- The From header has a new tag selected by the container; and the To header does not have a tag
- Record-Route and Via headers are not copied to the new request
- The Contact header is not copied unless the request is a REGISTER request

The container performs regular processing on the request such as adding the necessary Via headers when sending the new request.

Tip: This method provides the ability for a User Agent Client to respond to a 401 Unauthorized challenge while maintaining the Call-ID. This is useful for a client which needs to use the same Call-ID for a subsequent request, such as a client that is sending a registration request containing Authorization details.

9.3.5 Receiving responses

The SIP container receives responses which it may dispatch to SIP Servlets for handling.

Handling responses

When a SIP response is received by the SIP container, it selects the appropriate SIP Servlet to invoke. Based on the code for the response, it dispatches the response to one of the methods listed in Table 9-2.

Table 9-2 Response handling methods

Method	Response
doProvisionalResponse	1xx provisional responses
doSuccessResponse	2xx success responses
doRedirectResponse	3xx redirection responses
doErrorResponse	4xx, 5xx, 6xx bad request, server failure or global failure responses

An application that wishes to act on all SIP responses may override the `doResponse` method which is invoked for all SIP responses received by the SIP container.

Access to Headers, Message Content, and Transport Information are similarly available as receiving of SIP requests.

> **Note:** The `100 Trying` response and any retransmissions received by the SIP container are not dispatched to the `SipServlet`.

Sending ACK and CANCEL

The application is responsible for acknowledging all success (2xx) responses, while the container is responsible for acknowledging all non-2xx responses.

For an application to acknowledge a response, an ACK is created by invoking the `createAck()` method of the `SipServletRequest`. This can be modified further prior to being sent using the `send()` method of the `SipServletResponse`. Example 9-7 shows an example of acknowledging a final response.

Example 9-7 Acknowledging a success response

```
SipServletRequest request = response.getRequest();
request.createAck();
request.send();
```

9.3.6 Proxies

A common requirement for a SIP application is to proxy requests and responses.

Proxying Requests

In order to proxy a request, a `SipServlet` first obtains the `Proxy` object from the `SipServletRequest`. At the point of creation for the `Proxy` object the container will send a `100 Trying` response unless the `SipServlet` has already sent a 1xx response.

Once the `Proxy` object is obtained, the `SipServlet` can then invoke the `proxyTo(URI uri)` method to proxy the request to the specified `uri`. If the request is to be proxied to more than one destination, then the method `proxyTo(List listOfURIs)` is used instead.

Example 9-8 shows a simple proxy servlet which receives a request and proxies it to the Address of Record held in an internal store. If the Address of Record is not known by the proxy, it returns a `404 Not Found` response.

Example 9-8 SIP Servlet proxy example

```
public class LocalProxy extends javax.servlet.sip.SipServlet implements
javax.servlet.Servlet {

    protected void doRequest(SipServletRequest req) throws
ServletException, IOException {
```

```
        Proxy proxy = req.getProxy();
            Address contactAddress = (Address)
RegistrarStore.getContactForAOR(req.getTo().getURI());
            if ( contactAddress == null ) {
               req.createResponse(404);
               req.send();
            } else {
               proxy.proxyTo(contactAddress.getURI());
            }
        }
     }
 }
```

Controlling Proxy Operation

There are a number of controls over how the proxy operations are performed, they include the following:

- Record Route

  By default an application proxying a request will not remain on the application path. In order to see all subsequent requests in a dialog an application can set recordRoute to true.

- Supervised

  For a given transaction, the servlet will receive all responses by default. Where an application does not need to take explicit action on responses, the proxying of the responses received can be handled by the container without invoking the servlet. This is specified by setting the supervised parameter of the Proxy object to false.

- Parallel and sequential proxying

  When a SIP Servlet proxies a request to more than one destination, it may do so in parallel or sequentially. By default, proxying will be performed in parallel, however this may be changed by setting the parallel parameter of the Proxy object to false.

- Sequential search time-out

  When performing sequential proxying, the container waits for a period of time for a final response before it cancels the branch and moves on to proxy the next destination in the list. A default value may be provided using the <sequential-search-timeout> element of the SIP deployment descriptor. You can override the default setting or set the timeout on individual proxy objects using the sequentialSearchTimeout parameter of Proxy object.

- Stateful proxying

  By default, all proxying are stateful, however an application may perform stateless proxying by setting the `stateful` parameter of the `Proxy` object to `true`.

- Recursion

  By default, the container will automatically follow any redirect (3xx) responses received. Should an application not require this functionality, it can disabled it by setting the `recurse` parameter of the `Proxy` object to `false`.

9.3.7 Mapping requests to servlets

The SIP deployment descriptor provides a flexible way to declare the mapping between SIP requests and one or more SIP Servlets.

Mapping rules

The mapping rules specify the conditions under which servlets can be invoked. The rules language defines an object model for SIP requests. The model includes a number of types and associated properties. The Request object is the starting point for rules matching and it has the properties shown in Table 9-3.

Table 9-3 Request properties

Property	Description
method	the request method
uri	the request uri
from	the From header Address
to	the To header Address

The URI scheme is represented as URI object and can be accessed using the properties in Table 9-4 or Table 9-5 on page 220 depending on whether the URI is SipURI or TelURI, respectively.

Table 9-4 SipURI properties (extends URI)

Property	Description
scheme	either sip or sips
user	user part of the uri
host	host part of the uri
port	port number of the uri

Property	Description
tel	telephone number in the user part of the uri
param.*name*	value of named parameter within the uri

Table 9-5 TelURI properties (extends URI)

Property	Description
scheme	always tel
tel	subscriber name
param.*name*	value of named parameter within the uri

When constructing a matching rule, several different conditions which include operators and logical connectors are available. Table 9-6 shows the list of available operators.

Table 9-6 Operators

Condition	Description
equal	true if values are equal; false otherwise
exists	true if variable is defined; false otherwise
contains	true if variable contains the literal string; false otherwise
subdomain-of	true if the domain name or telephone subscriber is a subdomain of a literal value

Multiple rules may be combined using the boolean operators shown in Table 9-7.

Table 9-7 Connectors

Condition	Description
and	true if all conditions are true
or	true if any conditions are true
not	true if the condition is false

The sample mapping rule in Example 9-9 on page 221 includes two individual rules. The first rule tests the value of `request.method for equality to` "INVITE", and second rule will return true if the To header is a subdomain of ibm.com®. If a request is received where both of these conditions are true, then the `CallController` servlet will be invoked, subject to the ordering rules described in “Application composition” on page 221.

Example 9-9 Mapping Example

```
<servlet-mapping>
      <servlet-name>CallController</servlet-name>
      <pattern>
         <and>
            <equal>
               <var>request.method</var>
               <value>INVITE</value>
            </equal>
            <subdomain-of>
               <var>request.to.uri.host</var>
               <value>ibm.com</value>
            </subdomain-of>
         </and>
      </pattern>
   </servlet-mapping>
```

Application composition

Application composition depends on the order of deployed applications and on the order of mapping rules within the deployment descriptor of each application. To ensure consistent behavior, the order in which servlets are invoked is determined as follows:

- For an initial incoming request, the SIP container tries each potential rule in the deployed application order, then in the order the rules are listed in an application’s deployment descriptor. Upon finding the nth match, the container then invokes the corresponding servlet.
- If the servlet needs to proxy the request, the container re-scans the rules searching for additional matches. Upon finding the (n+1)th match, the container invokes the corresponding servlet.
- Matching the request excludes any servlet in the same application as the previously invoked servlet. No servlet will be invoked twice for the same SIP request.

9.3.8 Sessions

SIP Servlets are stateless and contain no per SIP dialog or per SIP transaction data. In order to allow a SIP Servlet to process multiple messages that are related, SIP Servlet provides the `SipSession` interface which represents a point-to-point relationship between two user agents.

The `SipApplicationSession` links multiple of these point-to-point relationships to allow them to be managed as a single application. For example, a SIP Servlet may provide a service such as a third party call control application which involves multiple user agents and an HTTP user interface. In this case, it is important to have the ability to link these individual point-to-point relationships together to comprise an application.

SipSession

A SipSession represents a SIP dialog to a SIP Servlet. There are some differences between the life cycle of a SIP dialog and a SipSession. Principally, the differences are:

- All messages within the SIP Servlet API belong to a SipSession, while some SIP requests (such as OPTIONS and REGISTER) may exist outside a dialog
- A SipSession is created immediately and does not require a 1xx or 2xx response to be received in order to establish the SipSession
- When a SIP dialog is terminated, the corresponding SipSession is not destroyed, it must be explicitly invalidated or timed out

Adding attributes

An application may store information in the `SipSession` or the `SipApplicationSession` using the `setAttribute(String name, Object attribute)` method on either of these objects.

Removing attributes

Information store in the `SipSession` or `SipApplicationSession` can be retrieved using the `getAttribute(String name)` method. If changes are made to the retrieved information, then `setAttribute` method must be invoked after the change.

Iterating over the Sessions

To iterate over the Sessions contained by the `SipApplicationSession`, two methods are available:

- `getSessions()`

  This method returns an `Iterator` over all sessions contained in `SipApplicationSession`.

- `getSessions(String protocol)`

This method returns an `Iterator` over only sessions of a specific protocol type. Specifying “SIP” will return `SipSessions` objects `and` “HTTP” will return `HttpSession` objects.

Session Lifetime

A SipSession will timeout when explicitly invalidated using the `invalidate()` method or when the containing `SipApplicationSession` is destroyed. Hence, the lifetime of a `SipSession` is not linked to the lifetime of a SIP dialog.

A `SipApplicationSession` may also be explicitly invalidated using the `invalidate()` method. In order to ensure that all sessions will be eventually garbage collected even if not explicitly invalidated, the `SipApplicationSession` has an inactivity timeout. The timeout is specified for an application by setting the <session-timeout> in the SIP Deployment Descriptor, and if not set, the timeout defaults to one minute. If a message is not received within that timeout period for one of the contained `SipSessions`, then the `SipApplicationSession` can be destroyed by the container.

The lifetime of a `SipApplicationSession can` be extended for a specified number of minutes by an application using the `setExpires(int minutes)` method.

An application may receive advance notice of a `SipApplicationSession` expiration by registering for the `sessionExpired` callback provided by the `SipApplicationSessionListener` interface. The expiry time may then be further extended.

Where a `SipApplicationSession` contains a `HttpSession`, the lifetime for the `HttpSession` is set not to expire. The `HttpSession` will expire when explicitly invalidated, or when the containing `SipApplicationSession` is destroyed.

An application wishing to determine the last access time prior to the current request may use the `getLastAccessedTime()` method on either the `SipApplicationSession` or `SipSession`.

9.3.9 Listeners and events

A SIP application may register to listen for events in the application. In addition to the servlet context events defined in `javax.servlet`, SIP Servlet defines a number of events that an application may listen for. Table 9-8 on page 224 shows the list of Listeners and the different events that they can listen for.

Table 9-8 SIP Servlet Listener types

Listeners	Events monitored
SipApplicationSessionListener	Creation, destruction or timeout of a SipApplicationSession
SipSessionListener	Creation or destruction of a SipSession
SipSessionAttributeListener	Addition, removal or replacement of attributes in the SipSession
SipErrorListener	ACK or PRACK not received within timeout period
TimerListener	Firing of a Timer

A listener implements the required interface in order to be invoked when the appropriate event occurs. The container notifies all `<listener>` classes that are registered in the SIP deployment descriptor when the event occurs.

An example of a listener class and the corresponding entry in the deployment descriptor is shown in Example 9-10 and Example 9-11. The sample code in Example 9-10 shows a Listener which extends the lifetime of a SipApplicationSession based on a condition which is evaluated when the SipApplicationSession has expired.

Example 9-10 Example SipApplicationSessionListener

```
package com.itso.sg2427255;

import javax.servlet.sip.SipApplicationSessionListener;
import javax.servlet.sip.SipApplicationSessionEvent;

public class MyListener implements SipApplicationSessionListener {

   //perform appropriate action on session expiration, extend if needed
   public void sessionExpired(SipApplicationSessionEvent ev) {
      SipApplicationSession session = ev.getApplicationSession();

      if ( someTest == true ) {
         session.setExpires(minutesToExtend);
      }
   }
}
```

Example 9-11 Example Listener entry in Deployment Descriptor

```
<listener>
   <listener-class>com.itso.sg247255.MyListener</listener-class>
</listener>
```

9.3.10 Timers

SIP Servlet provides the ability for an application to schedule timers and be notified by the container when these timers expire. This is useful where an applications wishes to perform an operation on an established session at some point in the future. For example, an application may require an announcement to be triggered as a result of a user using a service longer than a fixed period of time.

Creating a Timer

A TimerService object is available to the application using the attribute named `javax.servlet.sip.TimerService` in the application's `ServletContext`.

A `ServletTimer` is created using the `createTimer` method on the `TimeService`. The application specifies the `SipApplicationSession` that the `ServletTimer` will be associated with, the information that should be delivered when the timer expires, the delay prior to invoking the timer and whether the timer should persist across application server restarts. The createTimer method signature is as follows;

```
ServletTimer createTimer(SipApplicationSession appSession, long delay, 
boolean isPersistent, java.io.Serializable info)
```

A repeating timer may also be used. This can be set to repeat on a fixed time interval, or a fixed time after the last time that the timer fired, taking into account any delays experienced when firing the timer on the previous execution such as garbage collection or other processes. The alternative method signature is as follows:

```
ServletTimer createTimer(SipApplicationSession appSession, long delay, 
boolean fixedDelay, boolean isPersistent, java.io.Serializable info)
```

Listening for the expiration of Timers

An application that wishes to be notified of the expiration of timers implements the `TimerListener` interface and overrides the `timeout(ServletTimer timer)` method to handle the expiry event. In addition, the class that implements the TimerListener interface is registered as a <listener-class> in the deployment descriptor as shown in Example 9-12.

Example 9-12 Example <listener> deployment descriptor configuration

```
<listener>
   <listener-class>com.itso.sg247255.Listener</listener-class>
</listener>
```

9.3.11 Security

By default, no authentication or authorization is performed by the SIP Servlet container prior to invoking a SIP Servlet. An application may take advantage of the underlying authentication and authorization services provided by the container by either declaring the policies that the container is to enforce, or by developing application logic which relies on the container to perform the authentication and authorization.

Declarative security

An application may externalize security rules by specifying them in the deployment descriptor. First, one or more servlets and SIP methods are grouped together in a resource collection. For each resource collection, the following constraints may be applied:

- Authorization constraints

  The user must belong to at least one of security roles in the collection in order for access to the SIP Servlet to be granted

- Authentication type

  Specifies whether the container should return a `401 Unauthorized` or a `407 Proxy Authentication Required` response when sending an authentication challenge

- User data constraints

  Describes the characteristics of the transport that requests must be received over

All of these constraints must be met in order for the servlet to be invoked, otherwise the container will return a `401 Unauthorized` or `407 Proxy Authentication Required` response.

Programmatic security

An application may need to implement authorization requirements which cannot be expressed solely with the use of declarative security. For example, an application may allow all users to invoke a SIP Servlet, but may want to perform an authorization decision based on the contents of the message received. SIP Servlet provides the ability to obtain the username or Principal associated with the current authenticated request, or whether the current user is a member of a given role using the following methods on the `SipServletMessage` interface:

- `String getRemoteUser()`
- `java.security.Principal getUserPrincipal()`
- `boolean isUserInRole(String role)`

9.3.12 Converged servlet

An application may deliver services through multiple interfaces, for example through an HTTP interface to provide a user interface, and through SIP to provide call signalling. To facilitate the development of applications that combine both HTTP and SIP interfaces, WebSphere Application Server provides `ConvergedServlet`. By using `ConvergedServlet`, an `HttpServlet` is able to share resources with one or more `SipServlet` to create a converged application.

Extending the converged servlet

A `ConvergedServlet` can obtain a `SipApplicationSession` object by invoking the method `getApplicationSession(boolean createNewSession)` on the `ConvergeHttpServletRequest` object. An application may then operate on the collection of sessions contained by the `SipApplicationSession` object.

A `ConvergedServlet` is also able to obtain a `SipFactory` from the `ServletContext` to allow initial requests to be created.

Example 9-13 shows an example of both of these techniques.

Example 9-13 Example ConvergedServlet

```
public class CallControl extends ConvergedServlet implements Servlet {

private static final long serialVersionUID = 1L;
private SipFactory sipFactory;
   public void init() throws ServletException {
   sipFactory = (SipFactory)
getServletContext().getAttribute(SipServlet.SIP_FACTORY);

      if (sipFactory == null){
         System.out.println("No SipFactory in context object");
}

   protected void doGet(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
```

```
        ConvergeHttpServletRequest creq = (ConvergeHttpServletRequest)
req;
        SipApplicationSession appSession =
(SipApplicationSession)creq.getApplicationSession(true);
        String state = (String)appSession.getAttribute("state");

    ...
```

Forwards and includes

A `ConvergedServlet` may forward or include a JSP™ or another servlet as a result of servlet execution. The usual way for a `HttpServlet` to perform this is using the `getRequestDispatcher(String url)` method on the `HttpServletRequest`. Within a `ConvergedServlet`, this method returns `null`, and so a named dispatcher is used instead.

To locate a JSP using a named dispatcher, the JSP is configured with a name in the Web deployment descriptor. The dispatcher is then obtained from the `ServletContext` using the `getNamedDispatcher(String name)` method. The Web deployment descriptor stores a mapping between JSPs and names, and it is this name that is used to dispatch the include or forward request to the specific JSP. Example 9-14 shows an example of forwarding to a named JSP as shown in a Web deployment descriptor in Example 9-15.

Example 9-14 Forwarding to a JSP using a named dispatcher

```
ServletContext context = getServletContext();
context.getNamedDispatcher("CallControlStatus").forward(req, resp);
```

Example 9-15 Naming a JSP within the Web deployment descriptor

```
<servlet>
   <servlet-name>CallControlStatus.jsp</servlet-name>
   <jsp-file>CallControlStatus</jsp-file>
</servlet>
```

9.4 Best practices

There are a number of best practices for developing SIP Servlet applications. Many of these best practices are not new to SIP Servlet. Rather, they are well known practices for J2EE or SIP applications that are also applicable to SIP Servlet applications.

9.4.1 Application layering

The principles of separation of concerns, and the advantages of layering applications is well known. Several specific considerations apply to SIP Servlet applications.

Separate business logic from SIP Servlet

When developing a SIP Servlet, the temptation is to embed all business logic inside the SIP Servlet. Indeed, this temptation is greater than for HTTP servlet given the greater number of potential life cycle conditions that a SIP Servlet may need to handle.

However a clear advantage for separating the business logic from the SIP Servlet is that the business logic is able to be reused regardless of the communication channels through which requests are received.

A Plain Old Java Object (POJO) may be suitable as the basis for layering. However, where a service requires transaction management, to be exposed remotely or declarative security framework, then a Stateless Session Bean facade is recommended to provide the layering.

Make use of a state machine

The SIP protocol has a well defined state machine. The SIP container shields SIP application developers from complexities of the protocol including the state machine. However, the use of a state machine is recommended both for designing an application and for developing the logic to be executed at runtime. This has benefits of providing a standard way for communicating the design of the application, as well as the structure and flow of business logic within the application.

Decompose application into reusable components

Functional decomposition of applications into reusable components is a well known design approach. The SIP Servlet support for Application Composition adds additional considerations for building application components.

For example, rather than embedding the functionality of a Registrar in every SIP application, Application Composition can be used to allow a single Registrar to be deployed, and each SIP application will rely on this function being available. This allows reusable components to be created, rather than overloading a single SIP Servlet with multiple independent sets of functionality.

Hence, when designing a service to be deployed, consider whether it would be appropriate to decomposed the service into a collection of servlets which rely on Application Composition to deliver the functionality.

9.4.2 Message processing

Several SIP specific considerations apply when developing SIP Servlets to process SIP messages for optimal system performance.

Optimize for response time

An application which is slow to respond may cause user dissatisfaction, for example, taking longer than expected to establish a new call. In addition, the SIP protocol defines a number of circumstances where a protocol level retransmission will occur in the event that a message has not been received by a User Agent. If a server does not respond in a timely manner, then retransmissions may cause additional server load which exacerbates the delay in response and further decreases user satisfaction.

There are a number of considerations that are applicable to SIP Servlet applications. These include:

- Sending final responses as quickly as possible to avoid retransmissions
- Acknowledging responses as quickly as possible, again to avoid retransmissions
- Minimizing garbage collection, for example, minimizing object creation and tuning the garbage collector for latency over throughput

9.4.3 Implement specification design requirements

Applications that deliver functionality described in the SIP specification should consider associated best practices. Two areas that require carefully consideration are third party call control and registrar facility.

Third party call control best practices

RFC 3725 describes best practices for third party call control. When designing an application that implements third party call control, the best practices should be followed to choose the most appropriate flow for implementation. For example, an alternate call signalling sequence is appropriate when one of the called parties is an automata such as a voicemail server when compared to parties that may not establish a session in a timely manner.

Registrar requirements

An application implementing Registrar functionality should adhere to the requirements in Section 10 of RFC 3261. It specifies in some detail the steps that a Registrar must take, such as storing the received Call-ID and CSeq values in order to ensure that a consistent registration status is maintained by the Registrar.

9.4.4 Runtime development considerations

When developing a SIP application, it is important to consider the runtime environment that the application will be deployed in.

Threadsafety

A SIP Servlet may be invoked by concurrent threads. Static or instance variables in a SIP Servlet are generally discouraged, however where they are used, care should be taken to ensure that any access to these resources are in a threadsafe manner. In addition, when an application accesses a resource in a `SipSession` or a `SipApplicationSession`, again care should be taken to ensure that these resources are only accessed in a threadsafe manner.

Design for distribution and failover

Where a SIP Servlet is deployed in a clustered environment, the application is marked as distributable in the deployment descriptor. The container will then ensure that all requests within a given SipApplicationSession are directed to the same application server. In order to allow the contents of the `SipApplicationSession` and `SipSession` to fail over to another application server, all attributes placed in these session must implement the `Serializable` interface.

In addition, an application sending an initial request should use the host and port of the Stateless SIP Proxy when sending request to a WebSphere Application Server cell. For example, when sending an INVITE, the request URI should be set of the host and port of the Stateless SIP Proxy tier. This allows responses to be directed via the Stateless SIP Proxy tier, rather than be directed only to a single Application Server, therefore preventing a response being lost in the event that an Application Server fails.

Given that static or instance variables are not distributed across Application Servers, where an application needs to share state that is not stored in the SipApplicationSession or SipSession, then an alternative approach must be used. Several alternatives are available in a WebSphere Application Server environment, such as using a database, using the DistributedMap or a distributed caching framework.

Minimize State

SipSessions consume a finite set of resources within the Application Server. In order to maximize scalability of an application, it is recommended that SipSessions are only created when needed. In addition, SipSessions should be explicitly invalidated when no longer in use. This allows the container to reclaim the resources and make them available to applications running with the container in a more timely manner.

10

Sample SIP applications

In this chapter we introduce two sample SIP Applications and describe how to use the WebSphere Application Server Toolkit to develop and test a simple SIP application and a converged SIP and HTTP application.

This chapter contains the following:

- Application overview
- Registrar and proxy application
- Third Party Call Control application

10.1 Application overview

This Chapter introduces you to two sample application which show different aspects of developing SIP applications using the WebSphere Application Server Toolkit (AST).

The first of these applications is a simple Registrar and Proxy application. It allows a User Agent to register an Address of Record with a Registrar Servlet. A second User Agent then sends an INVITE to a Proxy Servlet, which relies on the information stored by the registrar to proxy a request to the User Agent to facilitate a call between two softphones.

The second application demonstrates how to develop a Third Party Call Control Converged Application. A Web page is provided where two SIP URIs may be entered, which initiates a call between two softphones. The status of the call is updated on a Web page as it progresses through the call life cycle. We also demonstrate Application composition by combining the Registrar function from the first sample application with the Call Control function of this second application.

10.2 Registrar and proxy application

In this section, you will create, deploy and test a simple Registrar and Proxy application using AST.

10.2.1 The scenario

In this application, two softphone User Agents establish a VoIP session between each other. Each softphone periodically sends REGISTER messages to a Registrar servlet containing their current contact information.

Each User Agent's current contact information is then used by a LocalProxy servlet, so that when one of the User Agents sends an INVITE to the other using the LocalProxy servlet, the requests is proxied to the current location of the other User Agent.

The sequence diagram in Figure 10-1 on page 235 illustrated the interaction between two User Agents. The User Agents Alice and Bob first REGISTER with the Registrar servlet, then Alice places a call, sending an INVITE to Bob via the LocalProxy servlet. Once Bob answers the phone, a media session is established between the two parties.

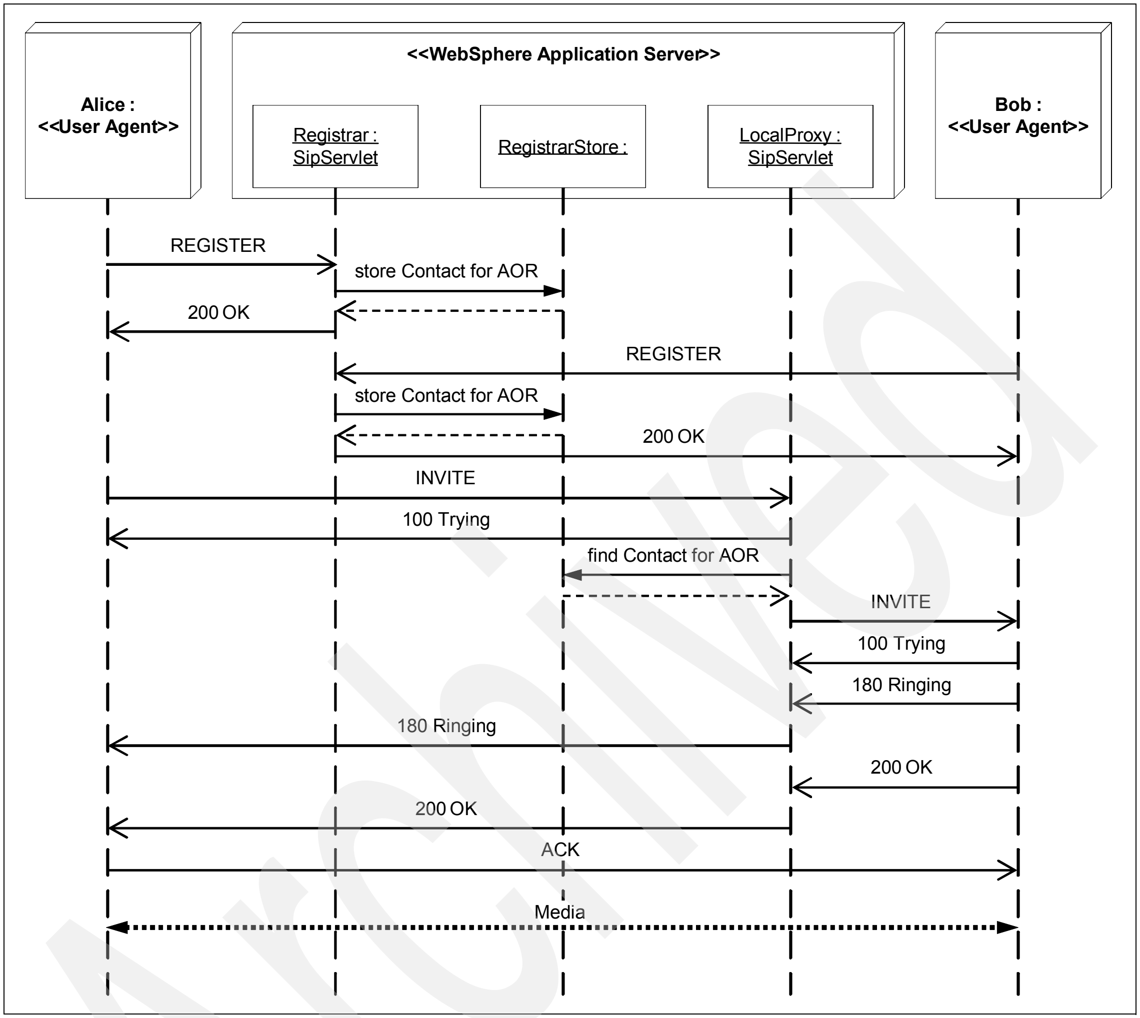


Figure 10-1 Sequence Diagram for Registrar and Proxy Sample

10.3 Creating the SIP application project

We will now create the SIP Application that contains the Registrar and LocalProxy SIP Servlets.

Note: The instructions for installing the WebSphere Application Server Toolkit and WebSphere Application Server can be found in Appendix A, “Installing the application development environment” on page 499 and Appendix B, “Installing the sample application test environment” on page 531.

Create the project

1. Launch AST by selecting **Start** → **All Programs** → **IBM WebSphere Application Server Toolkit V6.1** → **Application Server Toolkit**.
2. The Workspace Launcher will then load as shown in Figure 10-2.

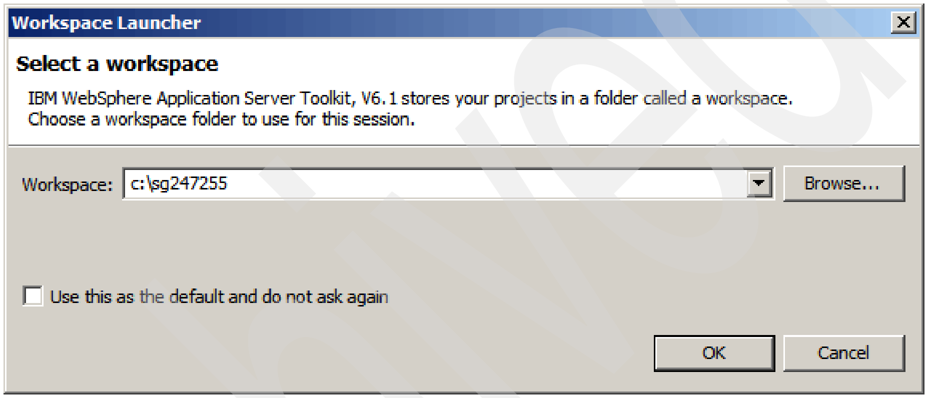


Figure 10-2 Selecting a workspace

3. Enter the path to your Workspace directory.
4. Click **OK** to continue.
5. Close the Welcome tab that is displayed on the Workspace.

   You are now ready to create a SIP project.
6. Select **File** → **New** → **Project**.

   The New Project Wizard will be displayed.
7. Select **SIP** → **SIP Project**.
8. Click **Next** to continue.

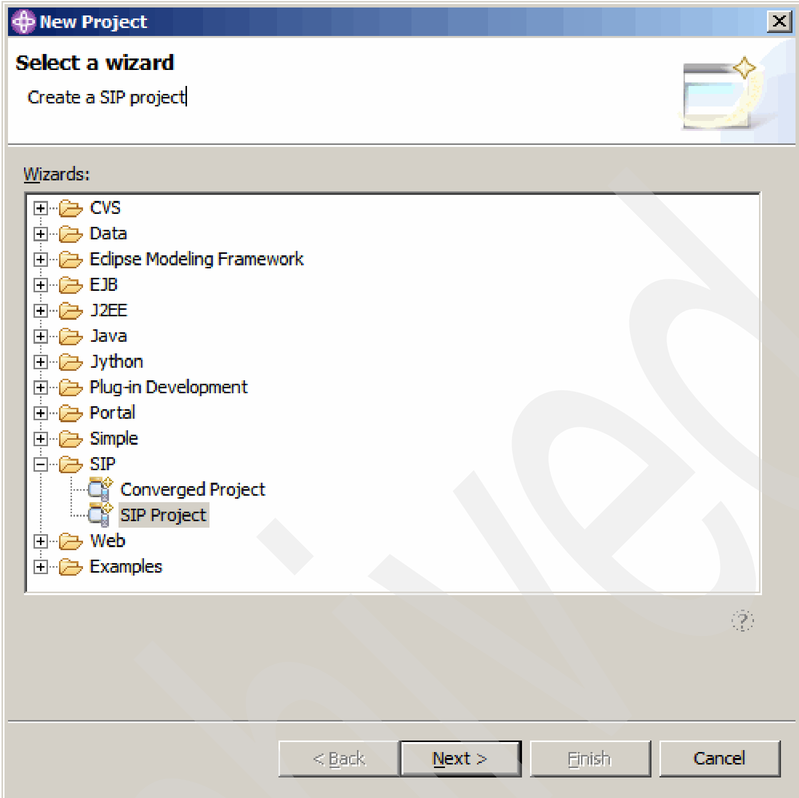


Figure 10-3 Create a SIP project - Select a Wizard

The New SIP Project dialog similar to Figure 10-4 on page 238 will be displayed.

9. Enter `RegistrarSample` for the Project Name.

10. Click **Next** to continue.

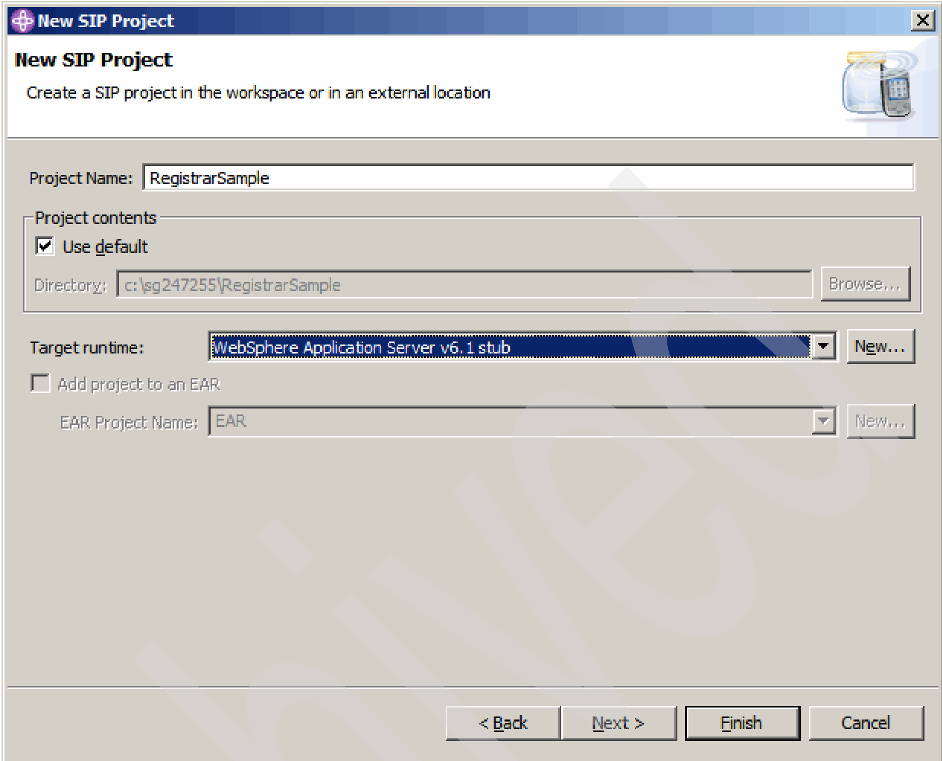


Figure 10-4 Specifying the Project Name, Contents and Target Runtime

A dialog showing the Project Facets will then be display.

11. Accept the default Project Facets and click **Finish** to create the new project.

12. Click **Yes** when prompted to switch to the J2EE perspective.

    The SIP Project has now been created and can now be inspected within the workspace.

13. Expand **Other Projects** → **RegistrarSample**.

    The SIP Toolkit has created the src folder, where the application code will be stored. In addition, a SIP deployment descriptor is created and added to the application.

10.3.1 Developing the SIP Servlets

In this section, we create the SIP Servlets for the Registrar and LocalProxy. We also create the business logic for storing mappings between Addresses of

Record provided by a User Agent and the User Agent's current contact information.

Create the Registrar SIP Servlet

To create the Registrar SIP servlet we will use AST.

1. Select **File** → **New** → **Other** to create the New Wizard.
2. Select **SIP** → **SIP Servlet**.
3. Click **Next**.

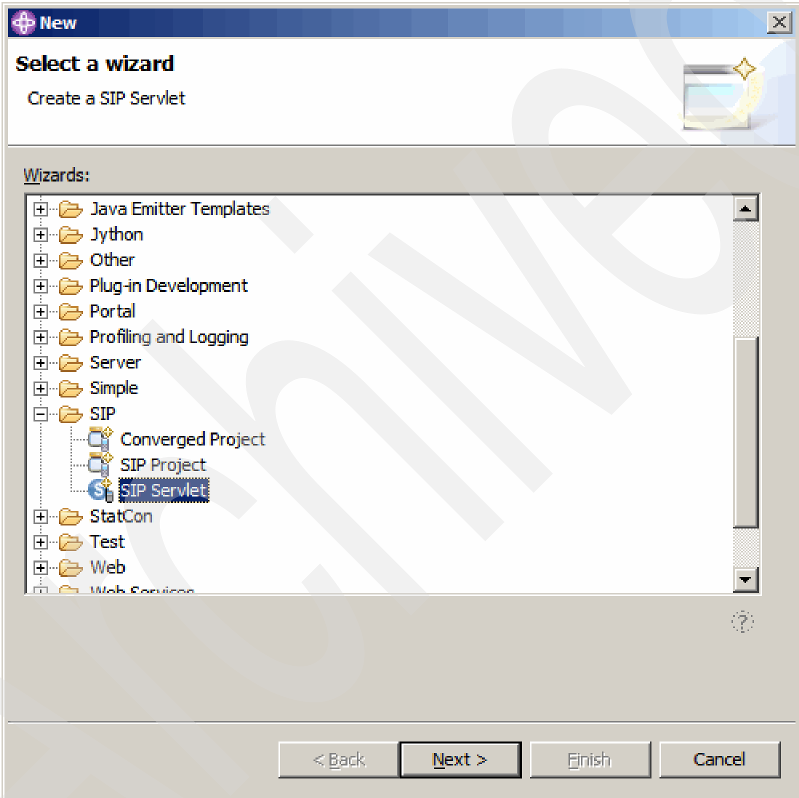


Figure 10-5 New SIP Servlet Wizard

4. The Create SIP Servlet: A Specify class file destination dialog, similar to Figure 10-6 on page 240, will be displayed.
   a. In the Java package field, enter: `com.ibm.itso.sg247255.sample1`
   b. In the Class name field, enter: `Registrar`
   c. Click **Next**.

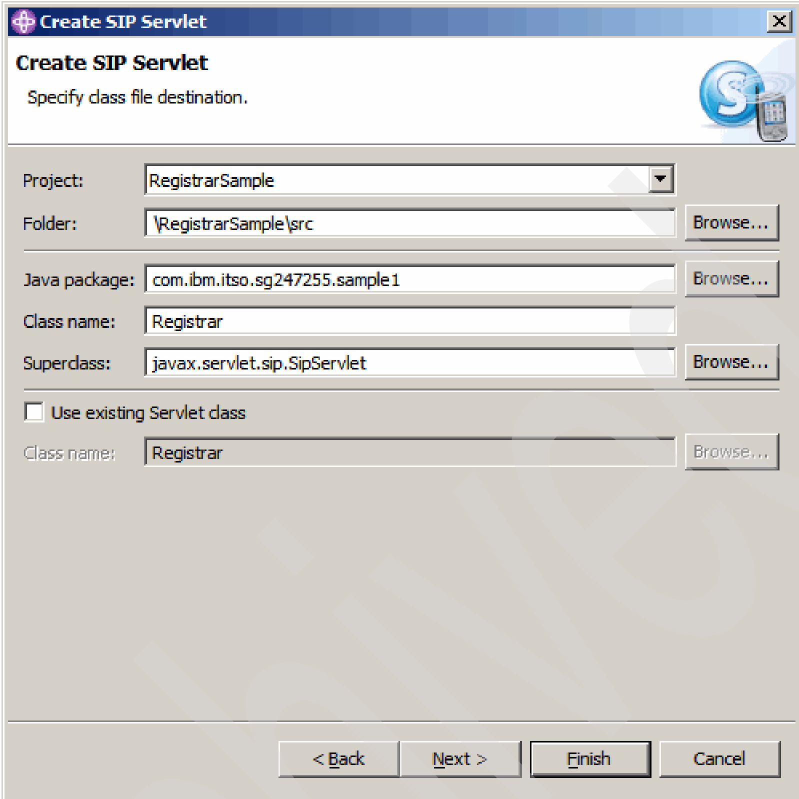


Figure 10-6 Create SIP Servlet - Specify class file destination

5. The Create SIP Servlet: The Enter servlet deployment descriptor specific information dialog, similar to Figure 10-7 on page 241, will be displayed. We need to add a Mapping Condition so that the Registrar servlet will receive requests that use the REGISTER method.
   a. Click **Add** to the right of the “Mappings” list.
   b. Then select **Condition**.

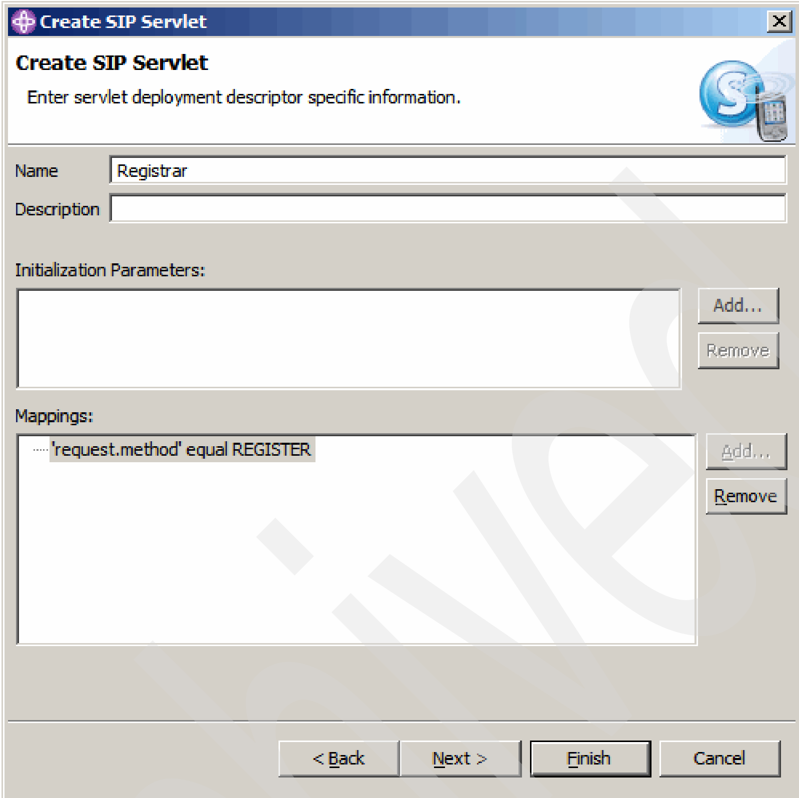


Figure 10-7 Create SIP Servlet - Enter servlet deployment descriptor specific information

6. The Add Mapping Condition dialog, similar to Figure 10-8 on page 242, will be displayed.
   a. In the Value field enter: `REGISTER`
   b. Click **OK**.
   c. Then click **Next**.

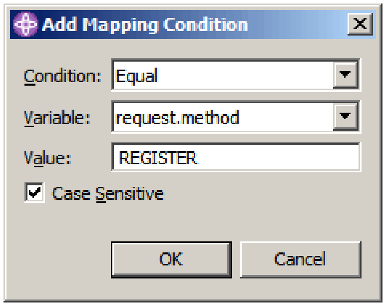


Figure 10-8 Add Mapping Condition

7. You can now specify the modifiers, interfaces to implement and method stubs to generate. For the Registrar sample, we will create the doRegister method stub.
   a. Click the **doRegister** check box.
   b. Click **Finish** to create the SIP Servlet.

Develop the Registrar business logic

The development structure of the Registrar business logic consist of two components:

- A `Registrar` SIP servlet

  It receives the REGISTER requests and invokes a local `RegistrationStore` class to store and/or remove the bindings between an Address of Record and a Contact.

- The `RegistrationStore`

  Stores the bindings in a simple in-memory Hashtable.

Registrar SIP servlet

Update the code generated for Registrar SipServlet with the code in Example 10-1.

Example 10-1 Registrar SipServlet

```
package com.ibm.itso.sg247255.sample1;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.servlet.Servlet;
import javax.servlet.ServletException;
```

```
import javax.servlet.sip.Address;
import javax.servlet.sip.SipServlet;
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipServletResponse;

public class Registrar extends SipServlet implements Servlet {

   private static final long serialVersionUID = -6178653233719984033L;
   static Logger logger =
Logger.getLogger("com.ibm.itso.sg247255.sample1.Registrar");

   protected void doRegister(SipServletRequest req) throws
ServletException, IOException {
      logger.log(Level.INFO, "Received REGISTER request for " +
req.getTo());
      Address AddressOfRecord = req.getAddressHeader("To");
      Address Contact = req.getAddressHeader("Contact");
      int Expires = req.getExpires();

      if ( Expires == 0 || Contact.getExpires() == 0) {
         RegistrarStore.removeContactForAOR(AddressOfRecord);
         req.createResponse(SipServletResponse.SC_OK).send();
      } else {
         RegistrarStore.updateContactForAOR(AddressOfRecord, Contact);
         SipServletResponse res = req.createResponse(200);
         res.addAddressHeader("Contact", Contact, false);
         res.send();
      }
   }
}
```

Registrar Store

The `Registrar` SIP servlet relies on the `RegistrarStore` to store the mapping between the Address of Record and Contact in an in-memory Hashtable.

Important: A production Registrar application would need to implement a number of additional features that are not included in this sample application.

For example, a production application would have additional business logic to handle multiple Contacts per Address of Record and would store the `CSeq` and `Call-ID` in order to correctly handle out of order requests. It would also store the bindings in a manner that could be distributed across servers, for example, using a relational database or distributed cache.

Refer to *RFC 3261, Section 10* for the requirements for developing a Registrar.

1. Expand **Other Projects** → **Registrar Sample** → **src**.
2. Right-click the package `com.ibm.itso.sg247255.sample1`.
3. Select **New** → **Class**.
4. In the field Name, enter `RegistrarStore`.
5. Click **Finish**.
6. Update the RegistrarStore class to use the code shown in Example 10-2.

Example 10-2 RegistrarStore code

```
package com.ibm.itso.sg247255.sample1;

import java.util.Hashtable;
import java.util.logging.*;

import javax.servlet.sip.Address;
import javax.servlet.sip.URI;

public class RegistrarStore {

   static Hashtable<URI, Address> Contacts = new Hashtable<URI,
Address>();
   static Logger logger =
Logger.getLogger("com.ibm.itso.sg247255.sample1.RegistrarStore");

   public static Address getContactForAOR ( Address AddressOfRecord ) {
      logger.log(Level.INFO, "Returning binding for AOR: " +
AddressOfRecord);
      return (Address) ( Contacts.get(AddressOfRecord.getURI()));
   }
```

```
    public static Address getContactForAOR ( URI URI ) {
        logger.log(Level.INFO, "Returning binding for AOR: " + URI);
        return (Address) ( Contacts.get(URI));
    }

    public static void updateContactForAOR ( Address AddressOfRecord,
Address Contact ) {
        logger.log(Level.INFO, "Updating binding for AOR: " +
AddressOfRecord + ", Contact is: " + Contact);
        if ( Contacts.containsKey(AddressOfRecord.getURI())) {
            Contacts.remove(AddressOfRecord.getURI());
        }

        Contacts.put(AddressOfRecord.getURI(), Contact);
    }

    public static void removeContactForAOR ( Address AddressOfRecord ) {
        logger.log(Level.INFO, "Removing binding for AOR: " +
AddressOfRecord);
        Contacts.remove(AddressOfRecord.getURI());
    }

    public static Hashtable getAllContacts () {
        logger.log(Level.INFO, "All Contacts: " + Contacts.toString());
        return Contacts;
    }
}
```

Develop the Proxy

We will now create the LocalProxy SIP servlet that will be used to proxy INVITE requests received to requested User Agent.

1. Select **File** → **New** → **Other**.
2. Expand **SIP**.
3. Select **SIP Servlet**.
    a. For the Java package field, enter: `com.ibm.itso.sg247255.sample1`
    b. For the Class name field, enter: `LocalProxy`
    c. Click **Next**.

4. Add Mapping so that the LocalProxy only receives requests that are for method INVITE.
   a. Click **Add** next to the list of Mappings.
   b. Select **Condition**.
      i. For the field Value, use INVITE.
      ii. Click **OK**.
   c. Click **Next**.
5. Select the **doInvite** check box.
6. Click **Finish**.
7. Complete the business logic for the LocalProxy servlet using the code shown in Example 10-3.

Example 10-3 LocalProxy

```
package com.ibm.itso.sg247255.sample1;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.servlet.Servlet;
import javax.servlet.ServletException;
import javax.servlet.sip.Address;
import javax.servlet.sip.Proxy;
import javax.servlet.sip.SipServlet;
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipServletResponse;
import javax.servlet.sip.URI;

import com.ibm.itso.sg247255.sample1.RegistrarStore;

public class LocalProxy extends SipServlet implements Servlet {

   private static final long serialVersionUID = 4629937833502893124L;
   static Logger logger =
Logger.getLogger("com.ibm.itso.sg247255.sample1.LocalProxy");

   protected void doInvite(SipServletRequest req) throws
ServletException, IOException {
      logger.log(Level.INFO, "Proxying INVITE received for " +
req.getTo());
      Proxy proxy = req.getProxy();
```

```
      URI uri = req.getTo().getURI();
      Address contactAddress = (Address)
RegistrarStore.getContactForAOR(uri);

      if ( contactAddress == null ) {
         SipServletResponse res =
req.createResponse(SipServletResponse.SC_NOT_FOUND);
         res.send();
      } else {
         proxy.proxyTo(contactAddress.getURI());
      }
   }
}}
```

Deployment descriptor

AST automatically creates and updates the deployment descriptor when SIP Servlets are created.

You can review the contents of the deployment descriptor by clicking twice on the SIP deployment descriptor. A summary of the information available in deployment descriptor is displayed as shown in Figure 10-9 on page 248.

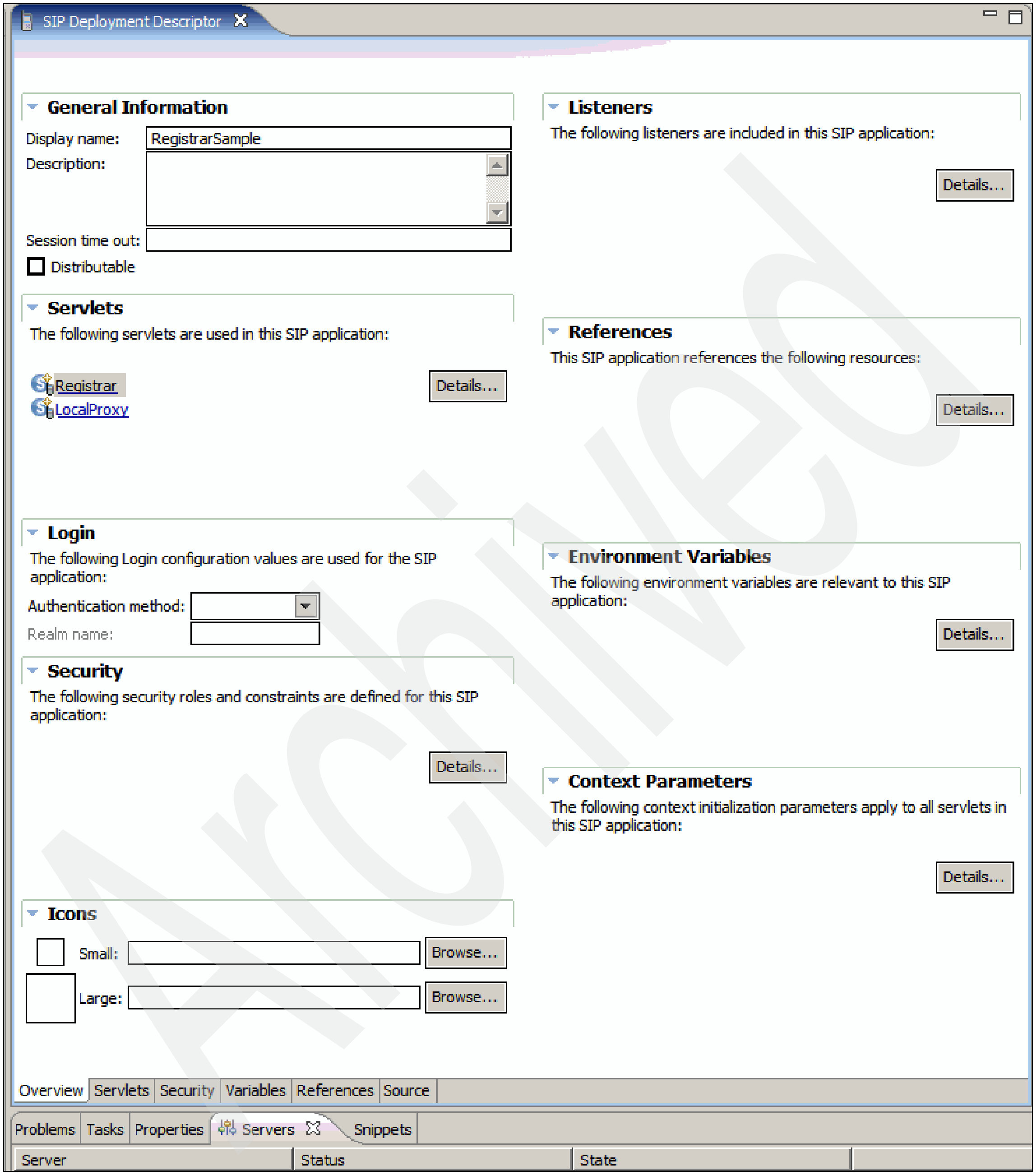


Figure 10-9 SIP Deployment Descriptor overview

To review the source for the SIP deployment descriptor, click the **Source** tab located at the bottom of the panel. The contents of the deployment descriptor are displayed as shown in Figure 10-10.

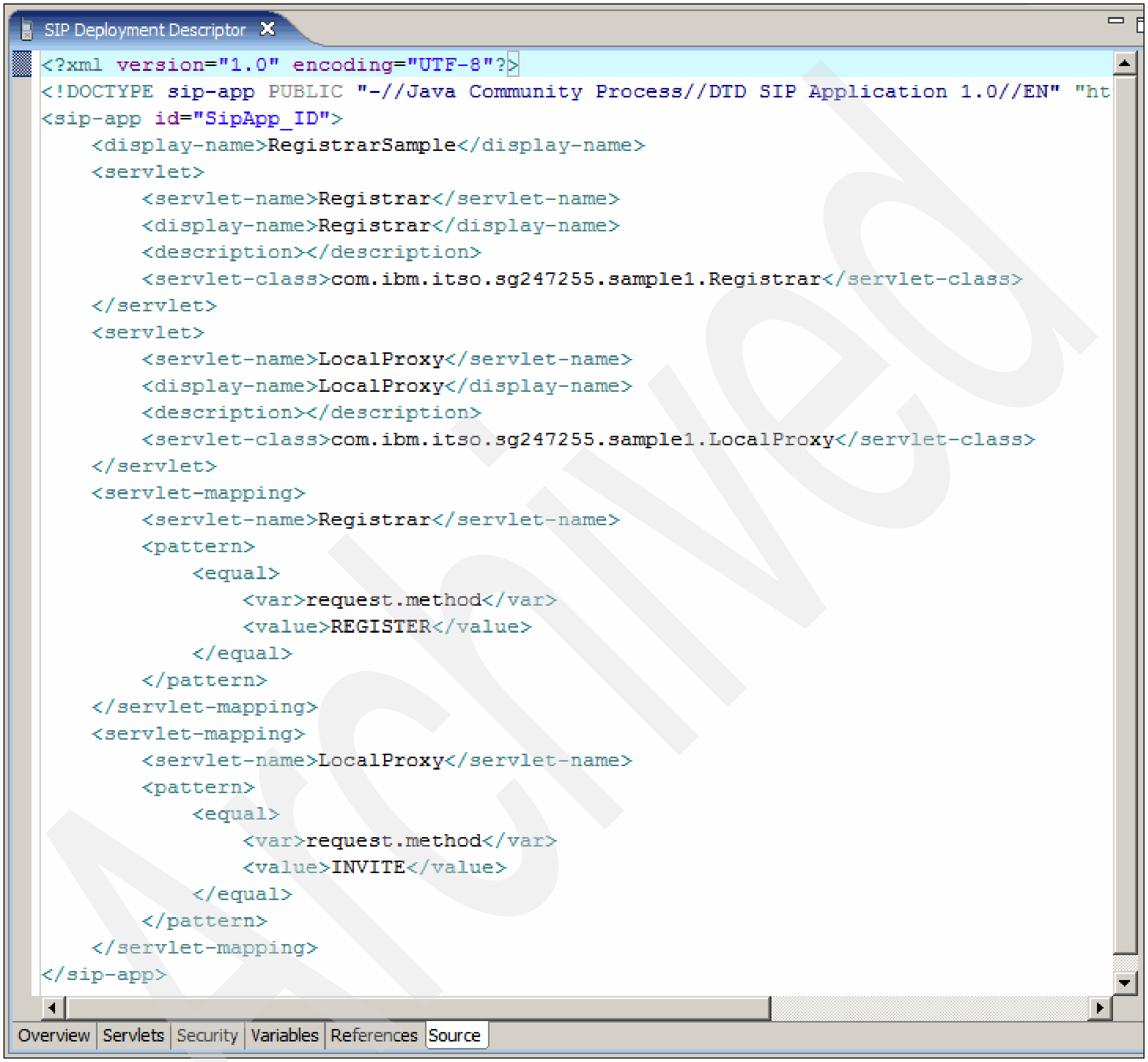


Figure 10-10 SIP Deployment Descriptor Source

Deploy the application using the Administration Console

To deploy your SIP application on the WebSphere Application Server you need to start the server. You also need to launch the Administration Console and enter your username and password.

8. Expand **Applications**.

9. Select **Install New Application**.

10. Enter the Full path of the of the application on the local file system.

11. In the Context root field, enter `RegistrarSample`.

    The Install New Application dialog will appear similar to Figure 10-11.

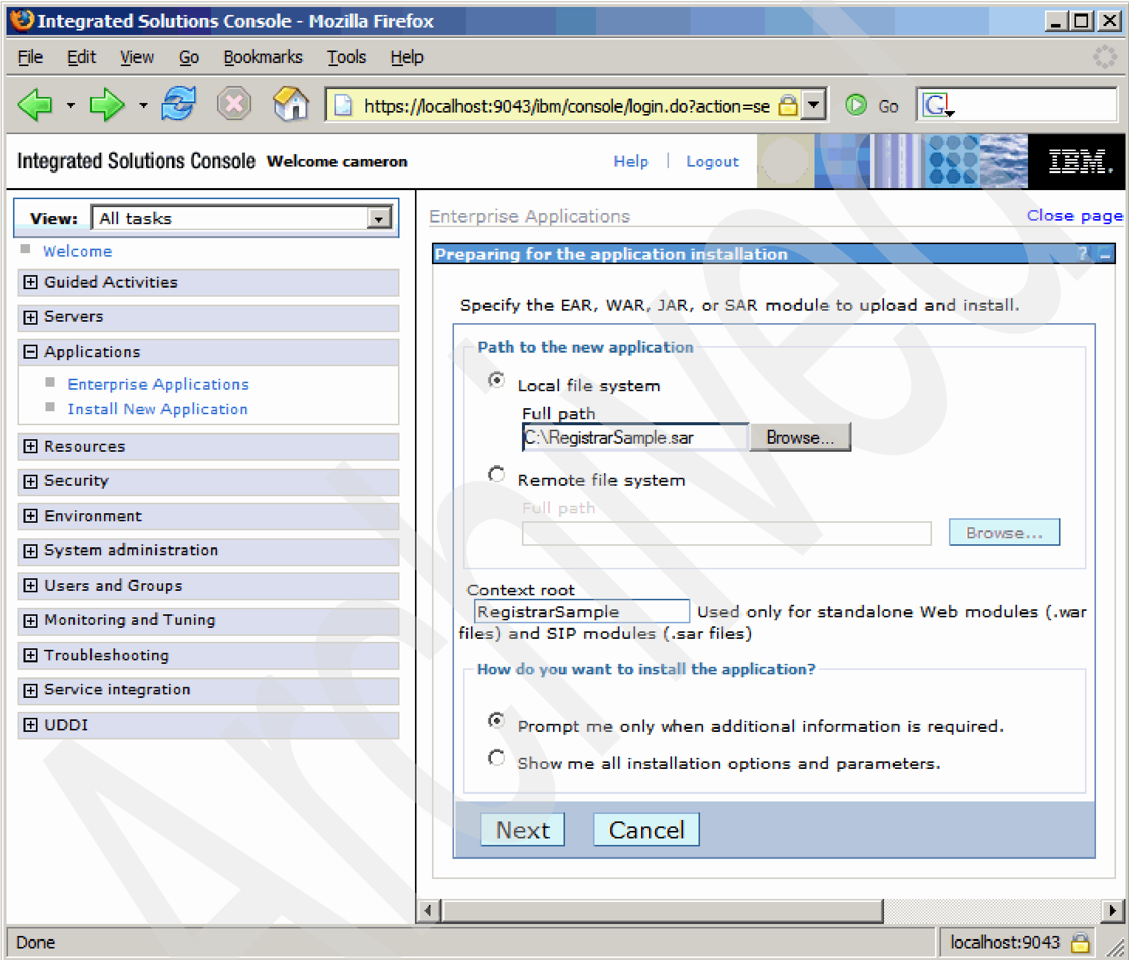


Figure 10-11 Installing the SIP Application Archive

12. Click **Next** to accept the default option of *Prompt me* only when additional information is required.

13. Click **Next** to accept the default installation options.

14. Click **Next** to accept the default module to server mapping.

15. Click **Next** to accept the default virtual host mapping.

16. Click **Finish**.
17. Click **Save** to save the application to the master configuration.
18. On the Enterprise Applications panel, select the **RegistrarSample_sar** application.
19. Click **Start** to start this application.

    The Administration Console will display that the application has been started as shown in Figure 10-12.

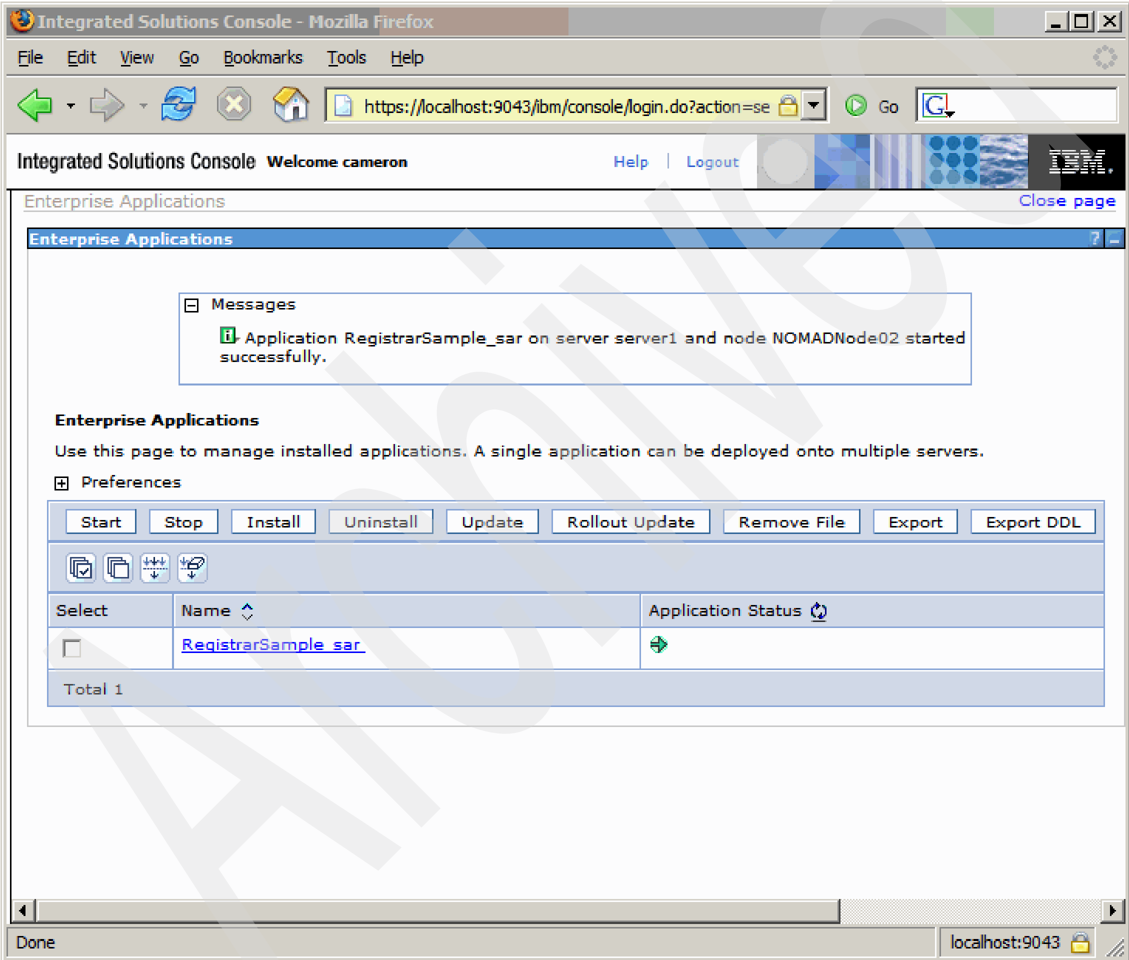


Figure 10-12 RegistrarSample started

10.3.2 Configure User Agents

We will use two softphone User Agents to test the applications. The softphones are installed on the same machine as WebSphere Application Server as a self contained unit test environment.

Note: The instructions for installing these User Agents can be found in A.2, “SIP device client installation” on page 502.

Two SIP User Agents are used, and the Address of Record for each User Agent is as follows:

- sip:alice@127.0.0.1
- sip:bob@127.0.0.1

Note: Using the loopback address of 127.0.0.1, simplifies the configuration for a unit test environment.

Configure X-Lite

The X-Lite softphone is configured as the first User Agent with the user account sip:alice@127.0.0.1

1. Click Add to add a new SIP Account.
2. On the Account tab, in the User Details box:
   a. For the Display Name field, enter: `Alice`
   b. For the User Name field, enter: `alice`
   c. For the Password field, enter: `alicepassword`
   d. For the Authorization user name field, enter: `alice`
   e. For the Domain field, enter: `127.0.0.1`
3. Click **OK**.
4. Click **Close** on the SIP Accounts dialog.

Configure SIPXPhone

The SIPXPhone is configured as the second user account, sip:bob@127.0.0.1. Locate the SIPXPhone installation directory.

1. Open the folder <sipxphone installation directory>\meta.
2. Open the file pinger-config in a text editor.

3. Locate the line starting with PHONESET_LINE.URL, change it to read as follows:

```
PHONESET_LINE.URL : sip:bob@127.0.0.1
```

4. Locate the line starting with SIP_TCP_PORT and change it to read as follows:

```
SIP_TCP_PORT : 5070
```

5. Locate the line starting with SIP_UDP_PORT and change it to read as follows:

```
SIP_UDP_PORT : 5070
```

6. Close and save the file

> **Note:** Configuring different ports, avoids port conflict between the User Agent and the WebSphere Application Server when you run them on the same machine.

10.3.3 Testing the Registrar and proxy application

We test the Registar and Proxy by launching both User Agents and completing a call between them.

Launch the User Agents

1. Launch X-Lite by Clicking **Start** → **All Programs** → **X-Lite** → **X-Lite**

   X-Lite will now load. Confirm that the panel shows the phrase `Your username is: alice` as shown in Figure 10-13 on page 254. This indicates that the User Agent successfully registered with the server.

Figure 10-13 X-Lite User Agents

2. Launch SIPXPhone by clicking **Start** → **All Programs** → **SipXFoundry** → **SipXPhone**

   SipXPhone will now load.Confirm that the SIP URI <sip:bob@127.0.0.1> is displayed in the top bar as shown in Figure 10-14 on page 255. This indicates that the User Agent has successfully registered with the server.

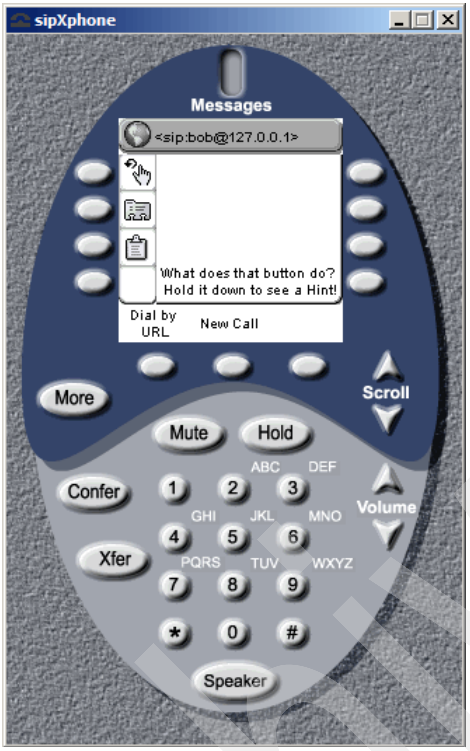


Figure 10-14 SIPXPhone User Agent

Place a call

We will now place a call between the two User Agents, relying on the Registrar and LocalProxy to locate the other User Agent.

> **Important:** Before answering the call, you may want to mute the microphone to minimize any feedback between the co-located User Agents.

1. In X-Lite, place a call by entering the URI of the other party. Enter sip:bob@127.0.0.1 and press **Enter**.
2. The X-Lite and sipXPhone User Agents will now show that a call is attempting to be established as shown in Figure 10-15 on page 256.

Figure 10-15 Call in Progress

3. Click **Answer** in SipXPhone to establish the call. Both User Agents will now have established a call
4. Complete the call by clicking **Disconnect** in sipXphone. This will terminate the call.

In this first application, we created a simple Registrar and Proxy SIP application using AST. We also deployed this application in WebSphere Application Server and tested it using two User Agents to establish a media session.

10.4 Third Party Call Control application

The second application introduces a Converged Application which will allow a third party to establish a call between two parties using a Web interface.

10.4.1 Overview

The Third Party Call Control application provides a Web interface where a user may initiate a call between two parties using a Web browser. The user initiates the call through the browser and tracks the progress of the call between the two parties. The functionality is delivered through a single Converged Application, combining both the Web interface for initiating the call and displaying the status, and the SIP interface for call signalling.

We demonstrate application composition between the Registrar and Proxy sample application developed in the previous section with the Third Party Call Control application. It allows the existing Registrar and Proxy services to be reused rather than reimplemented. It also demonstrates layered approach to building applications. For example, every application would not normally provide their own Registrar functionality, instead when required this would be a service deployed that all applications can rely on.

Figure 10-16 on page 258 shows the sequence diagram for a typical call being established.

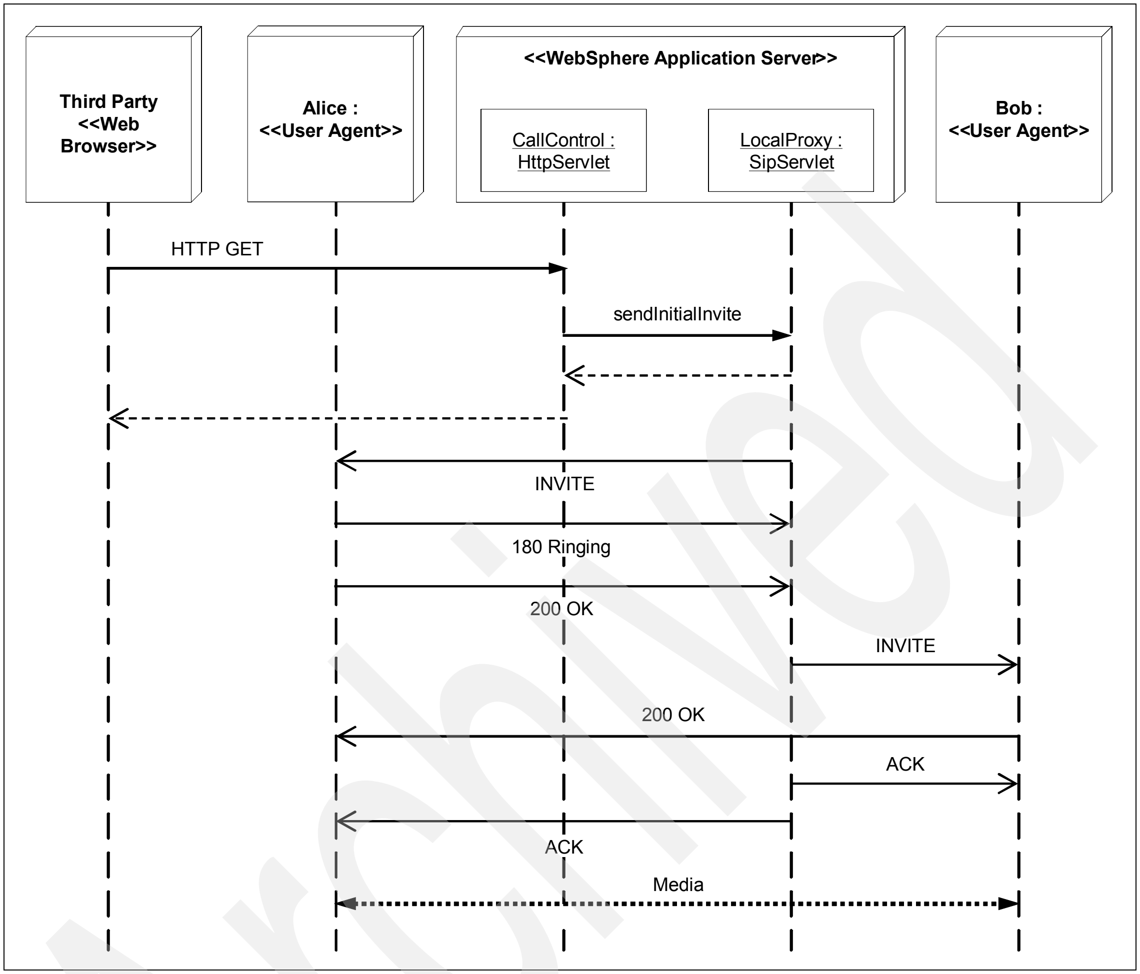


Figure 10-16 Sequence Diagram for Third Party Call Control application

10.4.2 Develop using the Application Server Toolkit

For this application, we will use the integrated development and debugging capabilities of the AST. We will also trace the execution for a detailed inspection of the request flow.

Create the Converged SIP project

The Third Party Call Control application uses both SIP and Web artifacts, so we will create a Converged Project which can contain both types of resources.

1. Select **File** → **New** → **SIP** → **Converged Project**

2. Click **Next**
3. In the Project Name field enter `ThirdPartyCallControl`
4. Click **Next**.
5. Accept the default Project Facets and Click **Next**
6. Accept the default Web Module settings and Click **Finish**

The ThirdPartyCallControl project will be created as shown in Figure 10-17.

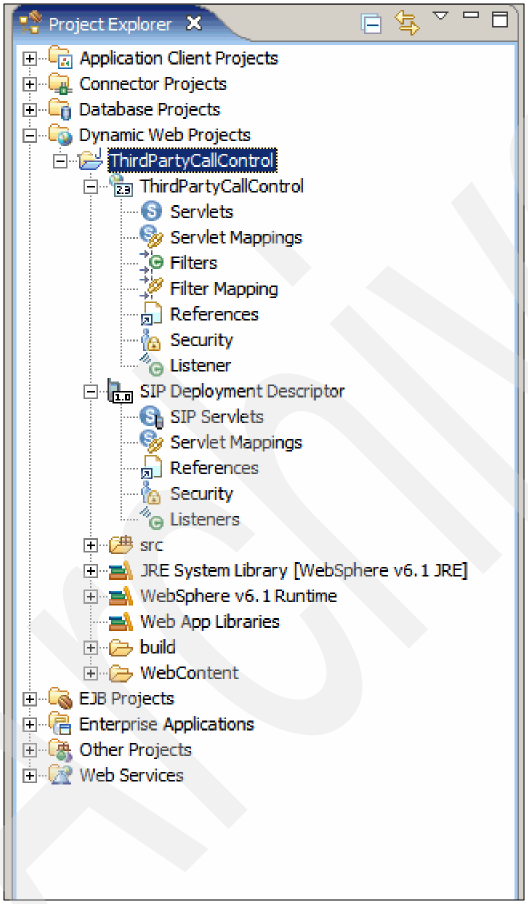


Figure 10-17 Third Party Call Control project

Create the CallControl servlet

We will now create the CallControl servlet which acts on commands received from the Web based interface for the application. We start by expanding the Dynamic Web Project and the ThirdPartyCallControl folder.

1. Right-click **ThirdPartyCallControl** folder.
2. Select **New** → **Servlet**. The Create Servlet dialog will be displayed.
3. In the Java package field, enter: `com.ibm.itso.sg247255.sample2`
4. In the Class name field, enter: `CallControl`
5. In the Superclass field, change the Superclass.
   a. Click **Browse**.
   b. Enter: `ConvergedServlet`.
   c. Select the Matching type:

      `ConvergedServlet - com.ibm.wsspi.sip.converge`
   d. The superclass field will now display: com.ibm.wsspi.sip.converge.ConvergedSevlet.
   e. Click **Next**.

   The Create Servlet dialog will appear similar to Figure 10-18.

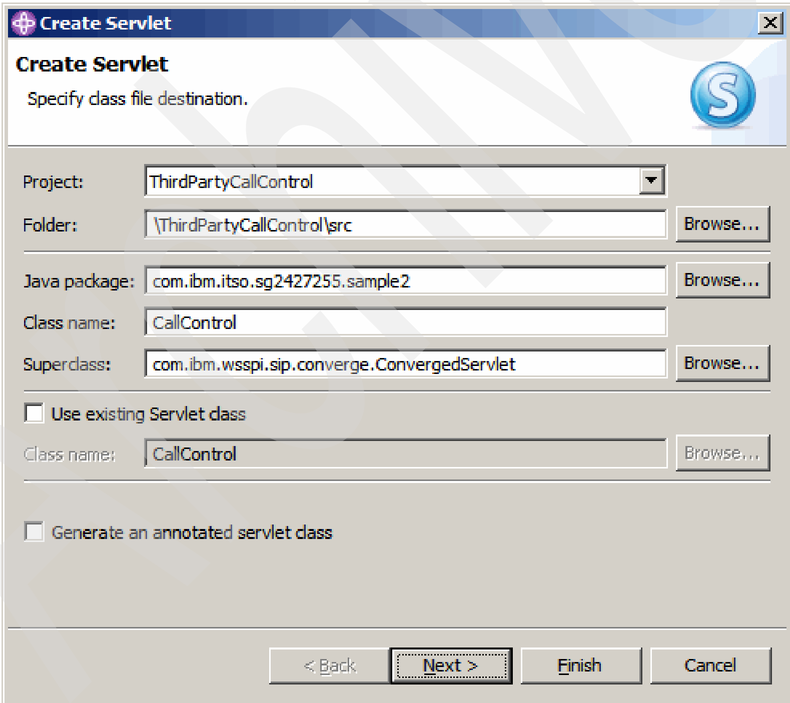


Figure 10-18 Create ClickToCall Servlet

6. Accept the default servlet deployment descriptor specific information and click **Next**.

The Create Servlet dialog will be displayed similar to Figure 10-19 on page 261.

7. Select the method stubs to be created.
   a. **Check** init.
   b. Confirm that the doGet is already checked.
   c. Deselect doPost.

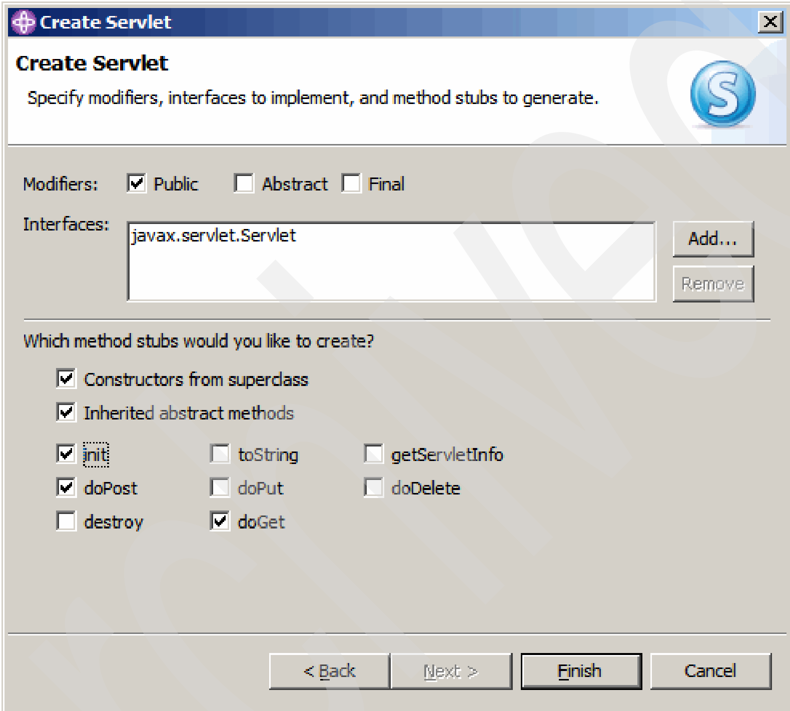


Figure 10-19 Specify modifiers, interfaces to implement, method stubs to generate

8. Click **Finish** to create the ConvergedServlet.
9. The CallControl servlet will now be created as shown in Figure 10-20 on page 262.

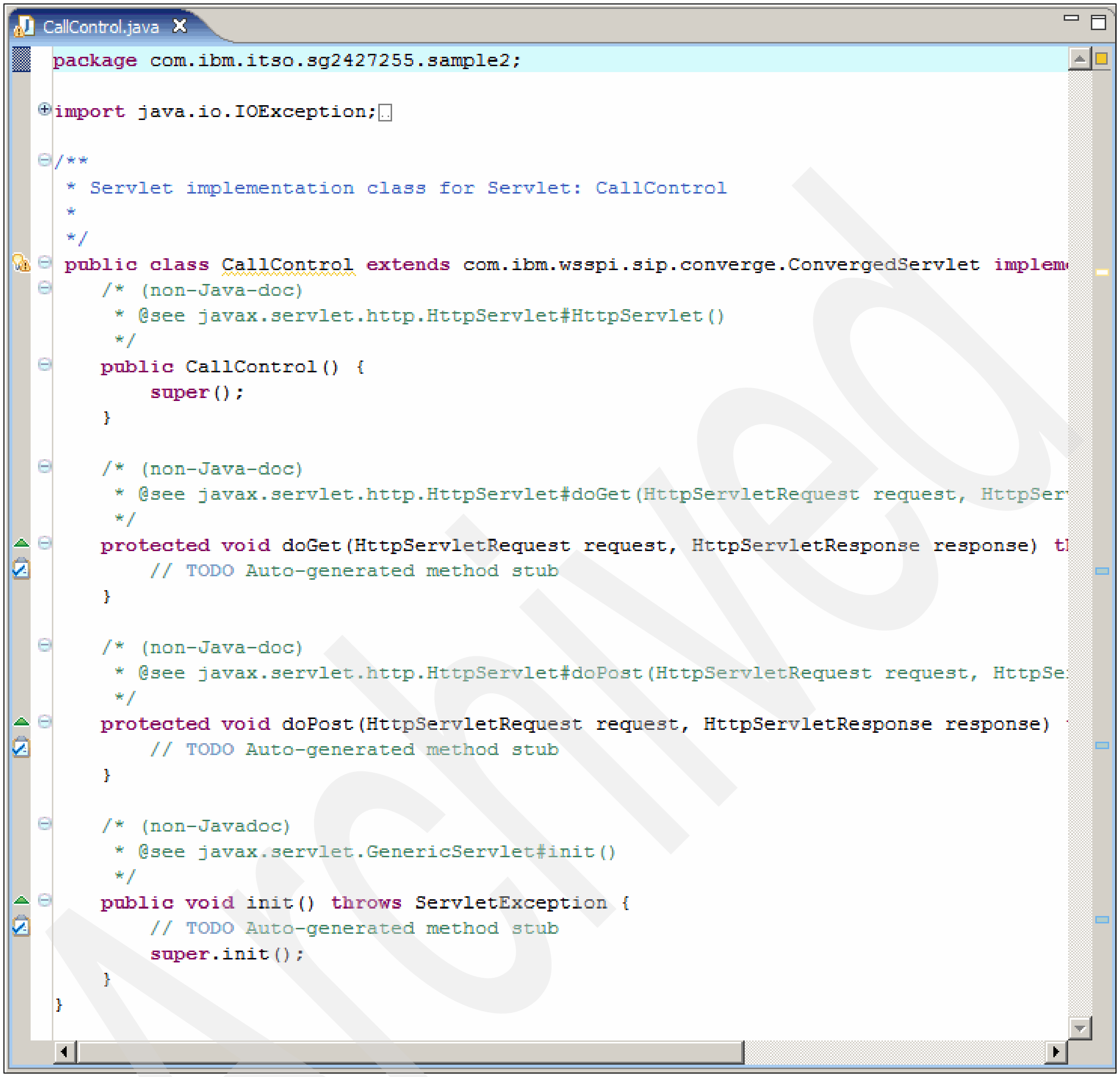


Figure 10-20 Generated Callcontrol Servlet

Complete the CallControl Business Logic

1. Update the CallControl servlet with the fields shown in Example 10-4.

Example 10-4 CallControl fields

```
private static final long serialVersionUID = -1939483920200031L;
private SipFactory sipFactory;
private static final String DEFAULT_PARTY_A = "sip:alice@127.0.0.1";
```

```
private static final String DEFAULT_PARTY_B = "sip:bob@127.0.0.1";
private static Logger logger =
Logger.getLogger("com.ibm.itso.sg247255.sample2.CallControl");
```

2. Update the CallControl servlet's init method as shown in Example 10-5.

Example 10-5 CallControl init method

```
public void init() throws ServletException {
   logger.log(Level.INFO, "ThirdPartyCCServer init");

   //obtain the sipFactory
   sipFactory = (SipFactory)
etServletContext().getAttribute(SipServlet.SIP_FACTORY);
   if (sipFactory == null){
      throw new ServletException("No SipFactory in context object");
   }
}
```

3. Update the CallControl servlet's doGet method as shown in Example 10-6.

Example 10-6 CallControl doGet method

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {

ConvergeHttpServletRequest creq = (ConvergeHttpServletRequest) req;
SipApplicationSession appSession =
(SipApplicationSession)creq.getApplicationSession(true);
   String state = (String)appSession.getAttribute("state");

SipURI toSipURI = null;
SipURI fromSipURI = null;
if ( req.getParameter("toSipURI") != null &&
req.getParameter("fromSipURI") != null ) {
   toSipURI =
(SipURI)sipFactory.createURI(req.getParameter("toSipURI"));
   fromSipURI =
(SipURI)sipFactory.createURI(req.getParameter("fromSipURI"));
} else {
   toSipURI = (SipURI)appSession.getAttribute("toSipURI");
   fromSipURI = (SipURI)appSession.getAttribute("fromSipURI");
}
//creating the session so we can always navigate back to the AppSession
HttpSession httpSession = req.getSession(true);
```

```
//if no call in progress, just include the JSP
if (state == null && req.getParameter("Establish") == null &&
req.getParameter("Cancel") == null) {
   req.setAttribute("pollInterval", 10000);
   req.setAttribute("toSipURI", DEFAULT_PARTY_A);
   req.setAttribute("fromSipURI", DEFAULT_PARTY_B);
   req.setAttribute("state", "No Call in Progress");
   req.setAttribute("createCallDisabled", "");
   req.setAttribute("terminateCallDisabled", "disabled");

//otherwise, we have a call in progress or have received a command to
perform
} else {

   req.setAttribute("toSipURI", toSipURI.toString() );
   req.setAttribute("fromSipURI", fromSipURI.toString() );

//if we've received a command to establish a call, send the first
INVITE
   if ( req.getParameter("Establish") != null) {
      CallController thirdPCC = (CallController)
getServletContext().getAttribute("callcontroller");
      thirdPCC.sendInitialInvite(appSession, fromSipURI, toSipURI);

      //disallow UI for creating new call while call is in progress
      req.setAttribute("createCallDisabled", "disabled");
      req.setAttribute("terminateCallDisabled", "");
   } else if (req.getParameter("Terminate") != null)  {
      //update status for callback
      CallController thirdPCC = (CallController)
getServletContext().getAttribute("callcontroller");
      thirdPCC.sendByeToAll(appSession);

      //disallow creating new or trying to terminate the call again
until terminate completed
      req.setAttribute("createCallDisabled", "disabled");
      req.setAttribute("terminateCallDisabled", "disabled");
   }
   //otherwise, just display status
   req.setAttribute("state", appSession.getAttribute("state"));

}

getServletContext().getNamedDispatcher("CallControlStatus.jsp").forward
(req, resp);
```

```
}
```

Create Call Control Status JSP

The Call Control Status JSP provides the Web user interface for creating and terminating the Third Party Call and reports on the status of the call. In this section we create the JSP.

Create the JSP

1. Expand Dynamic Web Projects → ThirdPartyCallControl → WebContent.
2. Click **New** → **JSP**.
3. Accept the default folder.
4. Enter a File name of `CallControlStatus.jsp`.
5. Click **Finish** to accept the default template.

Update the contents of the <body> as shown in Example 10-7.

Example 10-7 Call Control Status Body Content

```
<body onLoad="javascript:pollForStatus()">
<h1>Third Party Call Control Sample</h1>
<form name="CallControl" action="CallControl" method="get">
To : <input type="text" name ="toSipURI" value="<%=
request.getAttribute("toSipURI") %>"><br/>
From : <input type="text" name ="fromSipURI" value="<%=
request.getAttribute("fromSipURI") %>"><br/>
Call Status: <div id="state"><%= request.getAttribute("state")
%></div><br/>
<input type="submit" name="Establish" value="Establish" <%=
request.getAttribute("createCallDisabled") %>> <input type="submit"
name="Terminate" value="Terminate" <%=
request.getAttribute("terminateCallDisabled") %>><br/>
</form>
</body>
```

6. Update the contents of the <head> as shown in Example 10-8.

Example 10-8 Call Control Status Head content

```
<title>Third Party Call Control</title>
<script type="text/javascript">
var xmlHttp;
function createXMLHttpRequest() {
   if (window.ActiveXObject) {
      xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
```

```
   }
   else if (window.XMLHttpRequest) {
      xmlHttp = new XMLHttpRequest();
   }
}

function pollForStatus() {
   createXMLHttpRequest();
   var url = "CallControlStatusRPC";
   xmlHttp.open("GET", url, true);
   xmlHttp.onreadystatechange = pollForStatusCallback;
   xmlHttp.send(null);
}

function pollForStatusCallback() {
   if (xmlHttp.readyState == 4) {
      if (xmlHttp.status == 200) {
         //update the current call state
         var state =
xmlHttp.responseXML.getElementsByTagName("state")[0].firstChild.data;
         var elem = document.getElementById("state");
         elem.innerHTML = state;

         if ( state == "No Call in Progress") {
            document.forms[0].Establish.disabled = false;
            document.forms[0].Terminate.disabled = true;
         }

         setTimeout("pollForStatus()", 100);
      }
   }
}
</script>
</head>
```

Register the JSP Name

The CallControlStatus JSP will use a registered name in the deployment descriptor. This allows a NamedDispatcher to be used to dispatch to the JSP.

1. Expand Dynamic Web Projects.
2. Expand the ClickToCall folder.
3. Click twice on the ThirdPartyCallControl deployment descriptor. The deployment descriptor will be displayed as shown in Figure 10-21 on page 267.

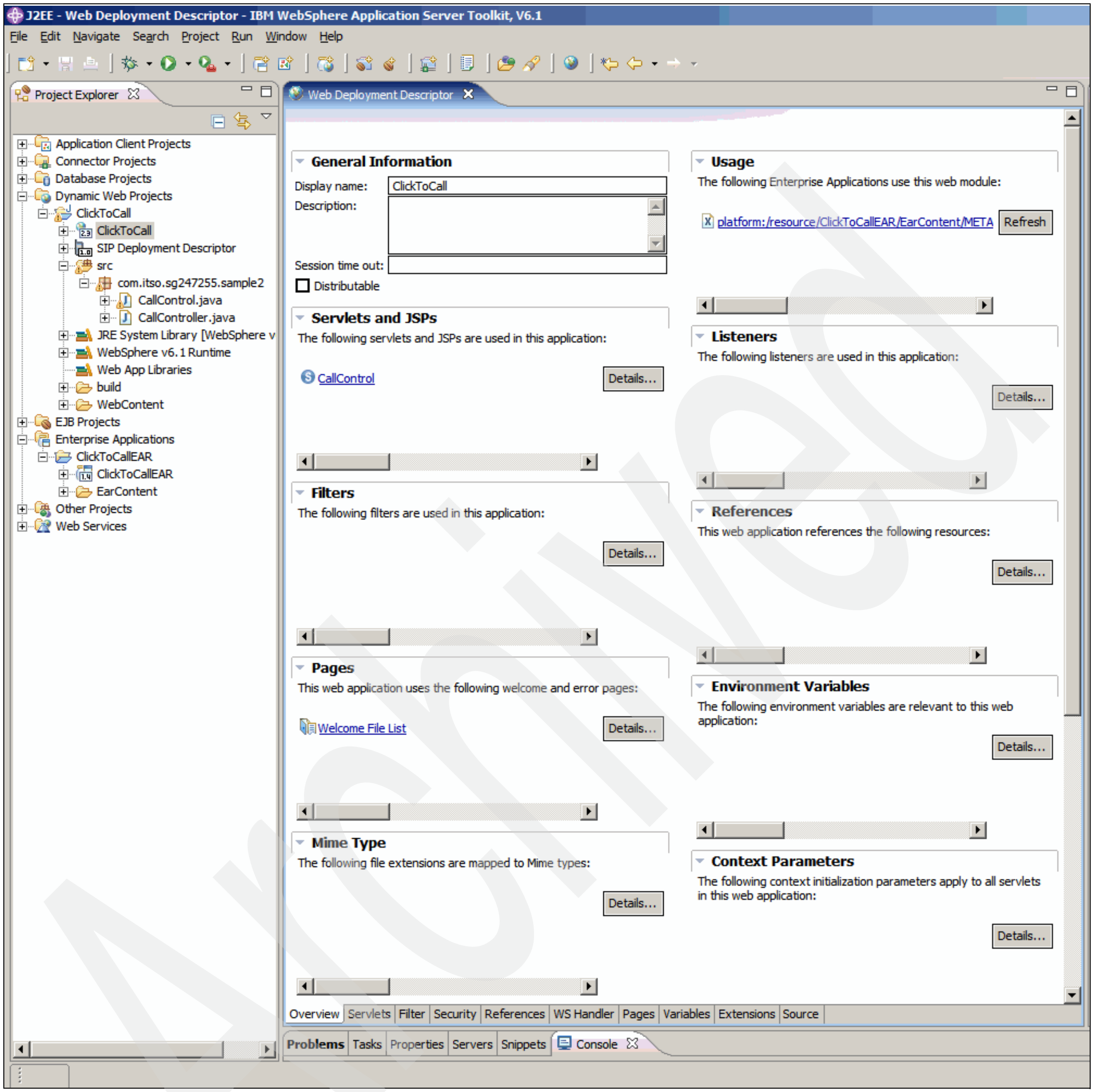


Figure 10-21 Web Deployment Descriptor

4. Click the Details button within the Servlets and JSPs section.
5. Edit the Web Deployment Descriptor to add the `<servlet>` stanza for the CallControlStatus.jsp.

Example 10-9 Web Deployment Descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app id="WebApp_ID">
   <display-name>ClickToCall</display-name>
   <servlet>
      <servlet-name>CallControl</servlet-name>
      <display-name>CallControl</display-name>
      <description></description>
      <servlet-class>
      com.ibm.itso.sg247255.sample2.CallControl</servlet-class>
   </servlet>
   <servlet>
      <servlet-name>CallControlStatus.jsp</servlet-name>
      <display-name>CallControlStatus.jsp</display-name>
      <description></description>
      <jsp-file>CallControlStatus.jsp</jsp-file>
   </servlet>
   <servlet-mapping>
      <servlet-name>CallControl</servlet-name>
      <url-pattern>/CallControl</url-pattern>
   </servlet-mapping>
   <welcome-file-list>
      <welcome-file>index.html</welcome-file>
      <welcome-file>index.htm</welcome-file>
      <welcome-file>index.jsp</welcome-file>
      <welcome-file>default.html</welcome-file>
      <welcome-file>default.htm</welcome-file>
      <welcome-file>default.jsp</welcome-file>
   </welcome-file-list>
</web-app>
```

Create CallControlStatusRPC Converged Servlet

1. Create a Converged Servlet named: `CallControlStatusRPC`

   (Follow the steps in “Create the CallControl servlet” on page 259.)

2. Update the CallControlStatusRPC servlet as shown in Example 10-10.

As the call progresses, SipApplicationSession is updated to allow the progress of the call to be monitored. The monitoring is performed using a ConvergedServlet. The ConvergedServlet relies on the link between an HttpSession and the SipApplicationSession.

Example 10-10 CallControlStatusRPC ConvergedServlet

```
package com.ibm.itso.sg2427255.sample2;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.ibm.websphere.servlet.session.IBMApplicationSession;
import com.ibm.wsspi.sip.converge.ConvergeHttpServletRequest;
import com.ibm.wsspi.sip.converge.ConvergedServlet;

public class CallControlStatusRPC extends ConvergedServlet implements
javax.servlet.Servlet {

   private static final long serialVersionUID = -49838594892L;
   static Logger logger =
Logger.getLogger("com.ibm.itso.sg247255.sample2.CallControlStatusRPC");

   protected void doGet(HttpServletRequest req, HttpServletResponse
res) throws ServletException, IOException {

      //obtain the ApplicationSession that this HttpSession is
contained by
      ConvergeHttpServletRequest convergedReq =
(ConvergeHttpServletRequest)req;
      IBMApplicationSession appSession =
convergedReq.getApplicationSession();

      //retrieve the state and pollInterval for this call
      String state = (String)appSession.getAttribute("state");

      //if no call in progress, initialise with defaults
      if ( state == null ) {
         state = new String("No Call in Progress");
      }
```

```
        //return to the AJAX client the current state of the call and
when next to poll
        PrintWriter out = res.getWriter();

        res.setContentType("text/xml");
        res.setHeader("Cache-Control", "no-cache");
        out.println("<state>" + state + "</state>");
        logger.log(Level.INFO, state);
        out.close();
    }

}
```

Create Call Control SipServlet

The Call Control SipServlet provides the necessary business logic for creating, managing and terminating the third party call.

1. Select **File** → **New** → **Other**.
2. Expand SIP and select **SIP Servlet** as shown in Figure 10-22.

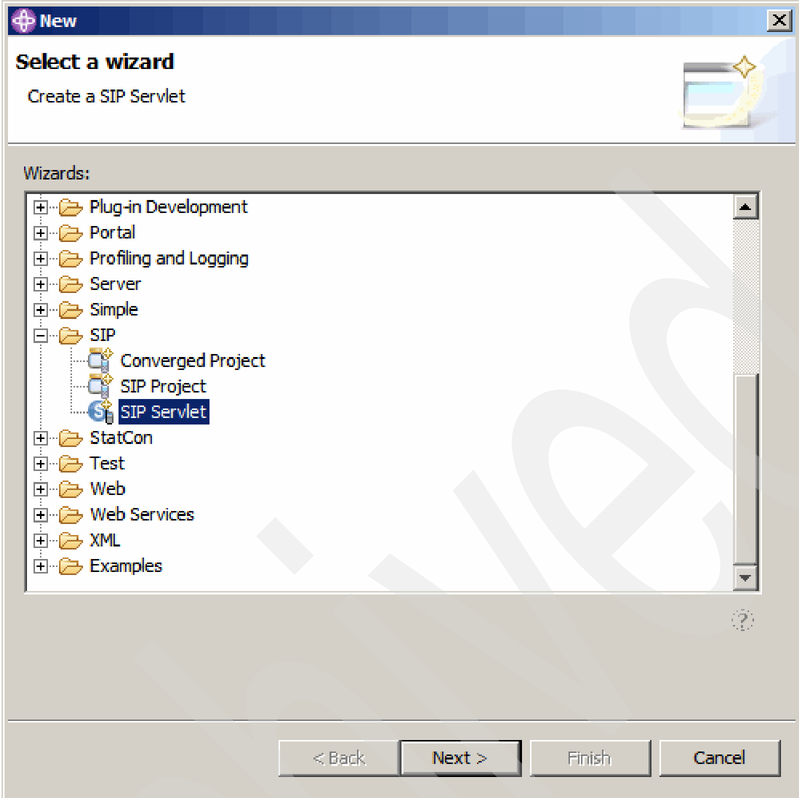


Figure 10-22 Create a SIP Servlet

The Create SIP Servlet dialog should appear as shown in Figure 10-23.

3. In the Java package field, enter: `com.ibm.itso.sg247255.sample2`
4. In the Class name field, enter: `CallControl`
5. Verify that the Superclass is: `javax.servlet.sip.SipServlet`
6. Click **Next**.

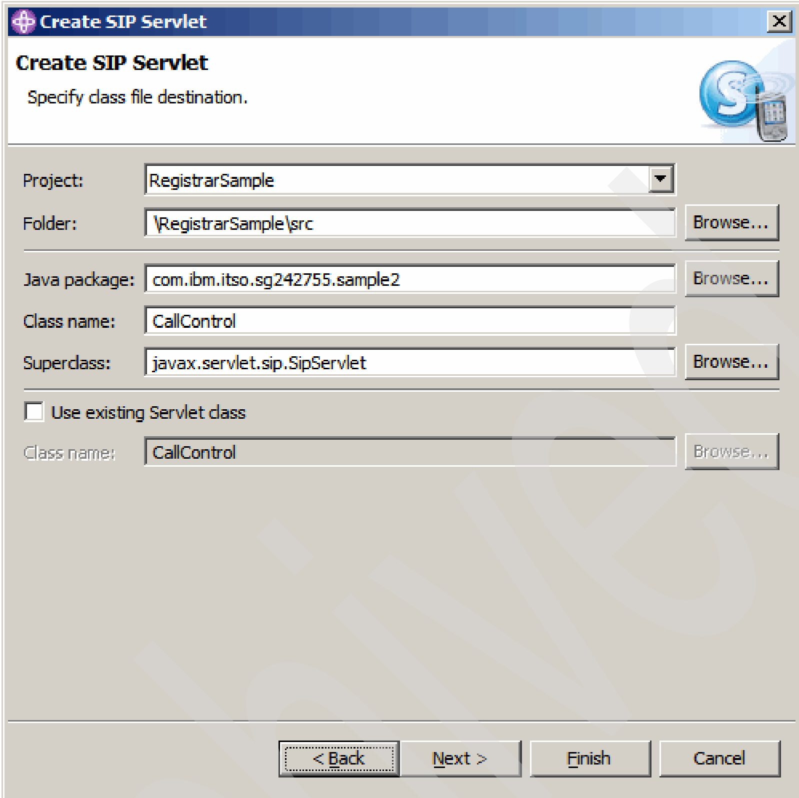


Figure 10-23 Create CallControl SipServlet

7. On the Servlet Deployment Descriptor Specific Information panel, accept the default and click **Next**.

> **Tip:** The CallControl SipServlet sends requests and receives the corresponding responses. The only Sip requests that it handles is the BYE request, which will only be sent as part of an existing Sip dialog.
>
> Hence, no mapping is required in the deployment descriptor for handling the BYE method. The container will dispatch the request to SipServlet that is part of the existing dialog.

The dialog for specifying modifiers, interfaces to implement and method stubs to generate will appear similar to Figure 10-24 on page 273.

8. Create method stubs for the CallControl SIP servlet.

   a. Check:

      `doBye, doErrorResponse, doProvisionalResponse, doSuccessResponse`

   b. Click **Finish** and the CallControl SIP servlet will be created.

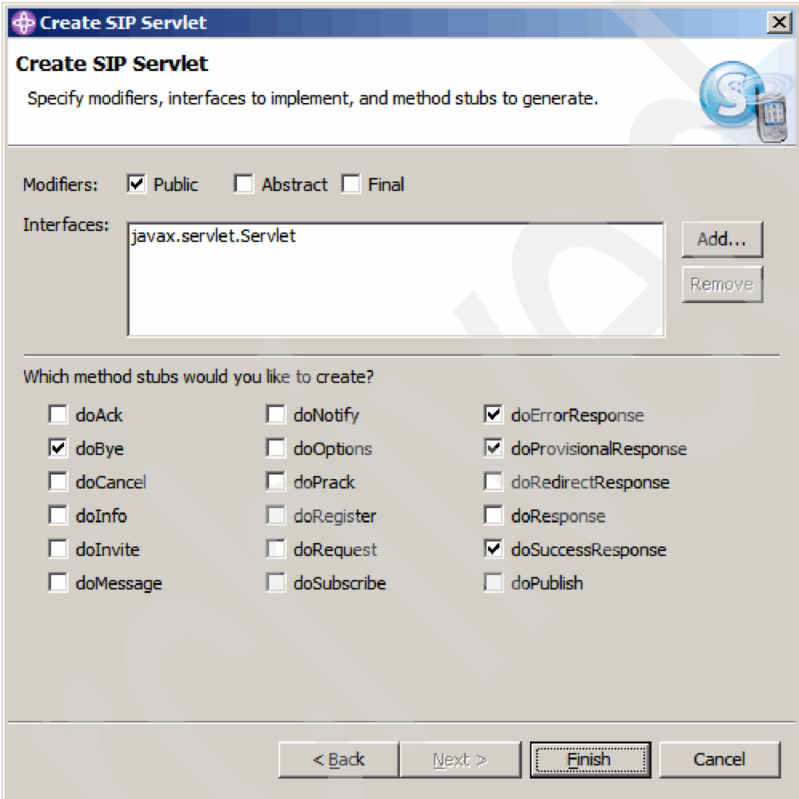


Figure 10-24 Specifying modifiers, interfaces to implement, method stubs to generate

9. Implement the doSuccessResponse method as shown in Example 10-11.

   The doSuccessResponse method is where the majority of the control logic lies. It is responsible for linking the two SIP dialogs with each other, so messages in one dialog may trigger messages in the other dialog. At call establishment time, the two dialogs are linked by storing the SipSession for the first dialog as a reference to the SipSession for the second dialog. This allows action to be taken on the opposing dialog. For example, if a BYE request is received in one dialog, the SipSession is used to locate the opposing dialog, so that the BYE request can be proxied on.

Example 10-11 CallController doSuccessResponse method

```
protected void doSuccessResponse(SipServletResponse resp) throws
IOException {
   logger.log(Level.INFO, "Received success response with method from "
+ resp.getFrom() );

   SipApplicationSession appSession = resp.getApplicationSession();

   //received a response to a BYE
   if (resp.getMethod().equalsIgnoreCase("BYE")) {
      logger.log(Level.INFO, "Terminated call with " +
resp.getFrom().getURI());
      appSession.setAttribute("state","Terminated call with " +
getUserFromAddress(resp.getFrom()));

      //if there are no more sessions, no dialogs, therefore sessions
are all terminated
      logger.log(Level.INFO, "Invalidating session " + resp.getFrom());
      resp.getSession().invalidate();
      Iterator it = appSession.getSessions("SIP");
      if ( !it.hasNext() ) {
         appSession.setAttribute("state","No Call in Progress");
      }
      return;
   }

   //received a response to something other than a BYE or an INVITE --
nothing to do
   if (!resp.getMethod().equalsIgnoreCase("INVITE")){
      return;
   }

   SipServletRequest req = resp.getRequest();
   SipServletRequest req2 = (SipServletRequest)
req.getAttribute("peer.req");

   SipURI to = (SipURI) req.getTo().getURI();
   SipURI from = (SipURI) req.getFrom().getURI();

   //if this is a response for the first invite
   if (req2 == null) {
      logger.log(Level.INFO, "Received response to the first INVITE");
```

```
      SipServletRequest invite =
sipFactory.createRequest(appSession,"INVITE",to,from);
      invite.setRequestURI(from);
      copyContent(resp, invite);
      invite.setAttribute("peer.req", req);
      invite.setAttribute("peer.resp", resp);

      //trigger session creation so we can relay subsequent requests
      SipSession dialog = resp.getSession();
      SipSession dialog2 = invite.getSession();

      // associate the two dialogs through the "peer" attribute
      dialog.setAttribute("peer", dialog2);
      dialog2.setAttribute("peer", dialog);

      req.getApplicationSession().setAttribute("state", new
String(to.getUser() + " answered, trying " + from.getUser() ));
      invite.send();

   } else {
      logger.log(Level.INFO, "Received response to the second INVITE");
      req.getApplicationSession().setAttribute("state", new
String("Call between " + from.getUser() + " and " + to.getUser() + "
established"));

      SipServletRequest ack = resp.createAck();
      ack.send();
      SipServletResponse peerResp =(SipServletResponse)
resp.getRequest().getAttribute("peer.resp");
      SipServletRequest ack2 = peerResp.createAck();
      copyContent(resp, ack2);
      ack2.send();
   }
}
```

10. Implement the doBye method as shown in Example 10-12.

    The doBye method is invoked when one of the User Agents terminates the call by sending a BYE request as part of the existing dialog.

Example 10-12 Call Controller doBye

```
   protected void doBye(SipServletRequest bye1) throws
ServletException, IOException {
      logger.log(Level.INFO, "Received a BYE from " + bye1.getFrom());
```

```
      if (bye1.isInitial()) {
         throw new ServletException("Got initial BYE from " +
getUserFromAddress(bye1.getFrom()));
      }

      SipSession dialog1 = bye1.getSession();
      SipSession dialog2 = (SipSession) dialog1.getAttribute("peer");

      if (dialog2 == null) {
         throw new ServletException("no peer SipSession; cannot forward
BYE");
      }

      SipServletRequest bye2 = dialog2.createRequest("BYE");
      bye2.setAttribute("peer.req", bye1);
      copyContent(bye1, bye2);
      bye1.getApplicationSession().setAttribute("state","Received BYE
from " + getUserFromAddress(bye1.getFrom()) + ", sending BYE to " +
getUserFromAddress(bye2.getTo()));
      bye1.createResponse(200);
      bye1.send();
      bye1.getSession().invalidate();
      Iterator it = bye1.getApplicationSession().getSessions("SIP");
      if ( !it.hasNext() ) {
         bye1.getApplicationSession().setAttribute("state","No Call in
Progress");
      }
      bye2.send();
```

11.Implement the sendInitialInvite method as shown in Example 10-13.

The sendInitialInvite method is invoked by the Call Control Servlet to initiate the call between the two parties, sending INVITE requests to the parties.

Example 10-13 Call Controller sendInitialInvite method

```
public void sendInitialInvite(SipApplicationSession appSession, SipURI
from, SipURI to) throws IOException {
   logger.log(Level.INFO, "Sending intial invite");
   SipServletRequest inviteReq =
sipFactory.createRequest(appSession,"INVITE",from, to);
   inviteReq.setRequestURI(to);
   SipSession session = inviteReq.getSession();
   try {
```

```
      session.setHandler("CallController");
   } catch (ServletException e) {
      // handle mismatch between deployment descriptor and code
   }

   appSession.setAttribute("toSipURI",to);
   appSession.setAttribute("fromSipURI",from);
   appSession.setAttribute("state", new String("Trying " + to.getUser()
));
   inviteReq.send();
}
```

12. Implement the init method as shown in Example 10-14.

    The init method is used to obtain a SipFactory from the ServletContext, and to store the CallController as an attribute in the ServletContext so this SipServlet may be located by other Servlets that want to initiate a Third Party Call.

Example 10-14 Call Controller init method

```
public void init() throws ServletException {
   logger.log(Level.INFO, "Initialising ThirdPartyCallController");
   getServletContext().setAttribute("callcontroller",this);

   sipFactory = (SipFactory)
getServletContext().getAttribute(SIP_FACTORY);
   if (sipFactory == null) {
      throw new ServletException("No SipFactory in context");
   }
}
```

13. In order for the init method to be invoked at server startup, load the SIP Servlet Deployment Descriptor
    a. Select the CallController servlet.
    b. Under the section Load on Startup, check the box Load on startup.
14. As this is a Converged Application, the same Servlet must be registered in the Web deployment descriptor.
    a. Click twice on the ThirdPartyCallControl deployment descriptor.
    b. Click Add to create a new Servlet definition.
    c. Check the box Use existing Servlet class.
    d. Click Browse.
    e. Select CallController.

f. Click Finish to create the servlet definition.

g. Under the section Load on Startup, check the box Load on startup.

15. Implement the sendByeToAll method as shown in Example 10-15 on page 278.

This method is used when the ThirdParty controller wants to terminate the established call. This sends a BYE to both parties of the dialog.

The two dialogs are located by iterating through the list of sessions of protocol type SIP contained in the SipApplicationSession. A given SipSession may be invalidated concurrently, a NullPointerException is explicitly used to catch and handle this situation.

Example 10-15 Call Controler sendByeToAll method

```
public void sendByeToAll(SipApplicationSession appSession) throws
IOException {
   logger.log(Level.INFO, "Terminating all calls");
   appSession.setAttribute("state","Terminating call");

   //send BYE on both dialogs
   Iterator iter = appSession.getSessions("SIP");
   while (iter.hasNext()) {
      logger.log(Level.INFO, "Sending a BYE for a session");
      try {
         SipSession aSession = (SipSession) iter.next();
         aSession.createRequest("BYE").send();
      } catch ( NullPointerException npe ) {
         logger.log(Level.INFO, "Failed to send BYE; session already
cleaned up");
      }
   }
```

10.4.3 Compose the Application

The application to be deployed will rely on the Registrar functionality developed in the previous sample. This allows the SIP URIs to be specified in the Call Control application in terms of well known Address of Records, rather than the more transient current contact information for a given user or device.

Rather that adding Registrar functionality directly to the application, this functionality shall be composed by combining the existing Registrar functionality with the newly developed Third Party Call Control functionality as shown in Figure 10-25 on page 279.

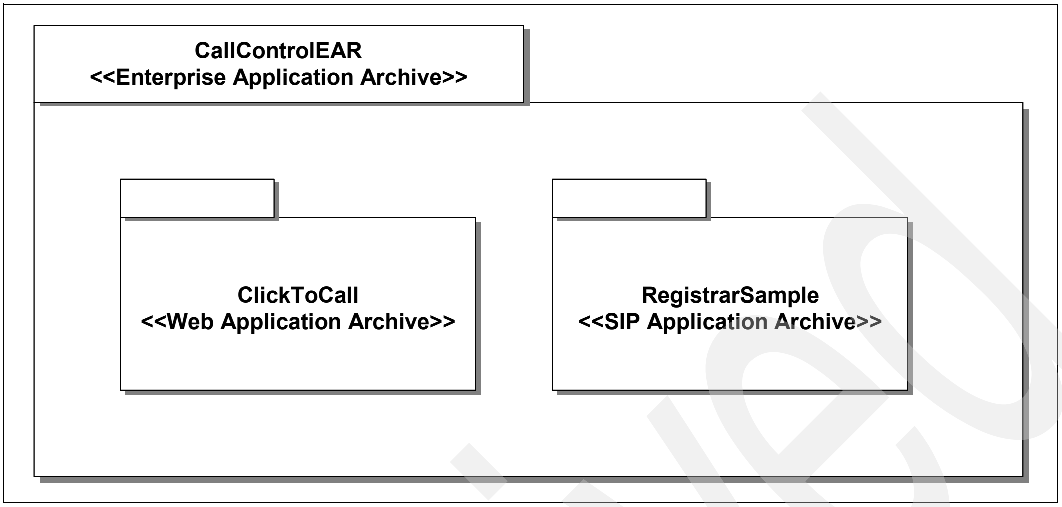


Figure 10-25 CallControlEAR

1. Select **File** → **New** → **Other**.
2. Expand the J2EE folder.
3. Click **Enterprise Application Project**.
4. Click **Next**.
5. Enter a Project Name of ThirdPartyCallControlEAR.
6. Click **Next**.
7. Accept the default Project Facets.
8. Click **Next**.
9. Add the ThirdPartyCallControl and RegistrarSample projects as Modules as shown in Figure 10-26 on page 280.

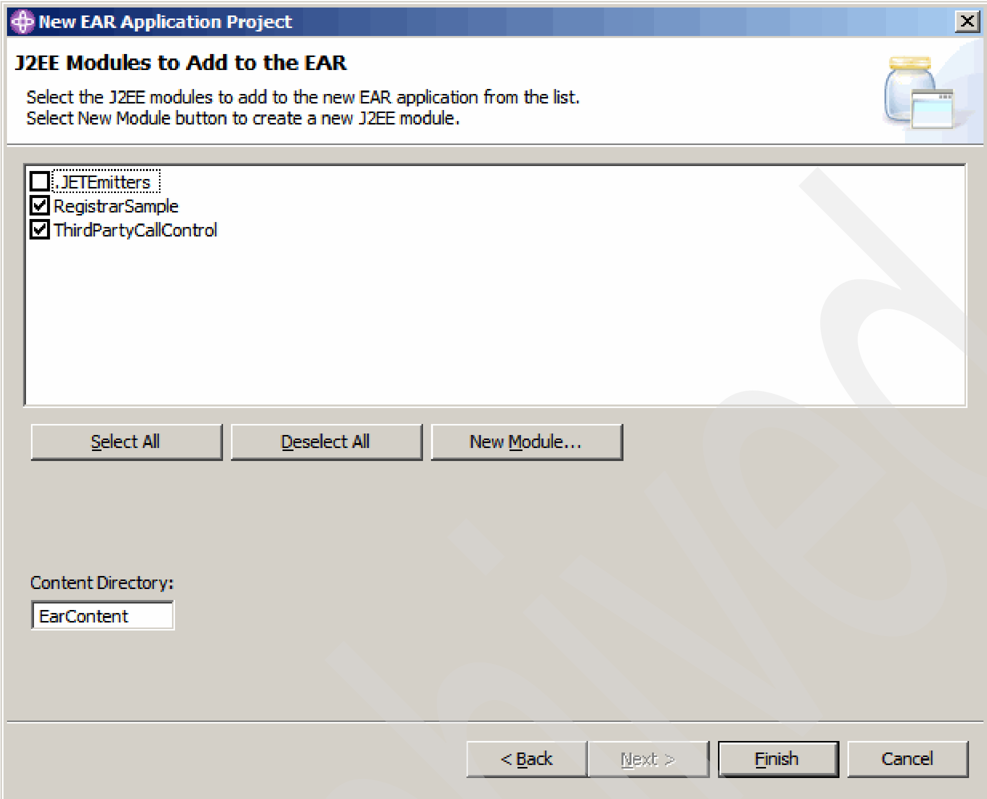


Figure 10-26 Add J2EE Modules to the EAR

10. Click **Finish** to create the EAR.

10.4.4 Deploy the converged SIP/J2EE application

If you have an application server instance already running from the previous sample, stop the instance. We will use this instance to automate the deployment of the application.

Configure the Application Server

1. Select **File** → **New** → **Other**.
2. Expand **Server**. Select **Server**.
3. Click **Next**.

   The New Server dialog will display as shown in Figure 10-27 on page 281.

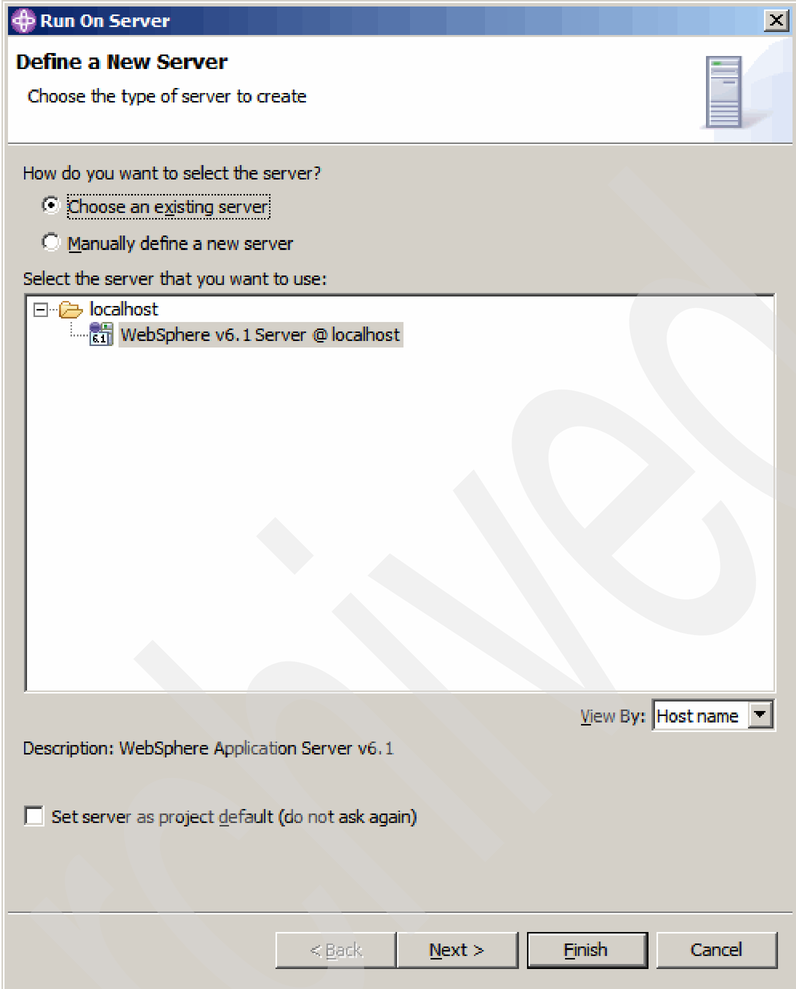


Figure 10-27 Define a New Server

4. Accept the defaults and click **Next**.

This will allow you to configure the Server installed earlier as the Server to publish the project to. The wizard will now locate the available profiles and display these in the WebSphere Server Settings dialog similar to that shown in Figure 10-28 on page 282.

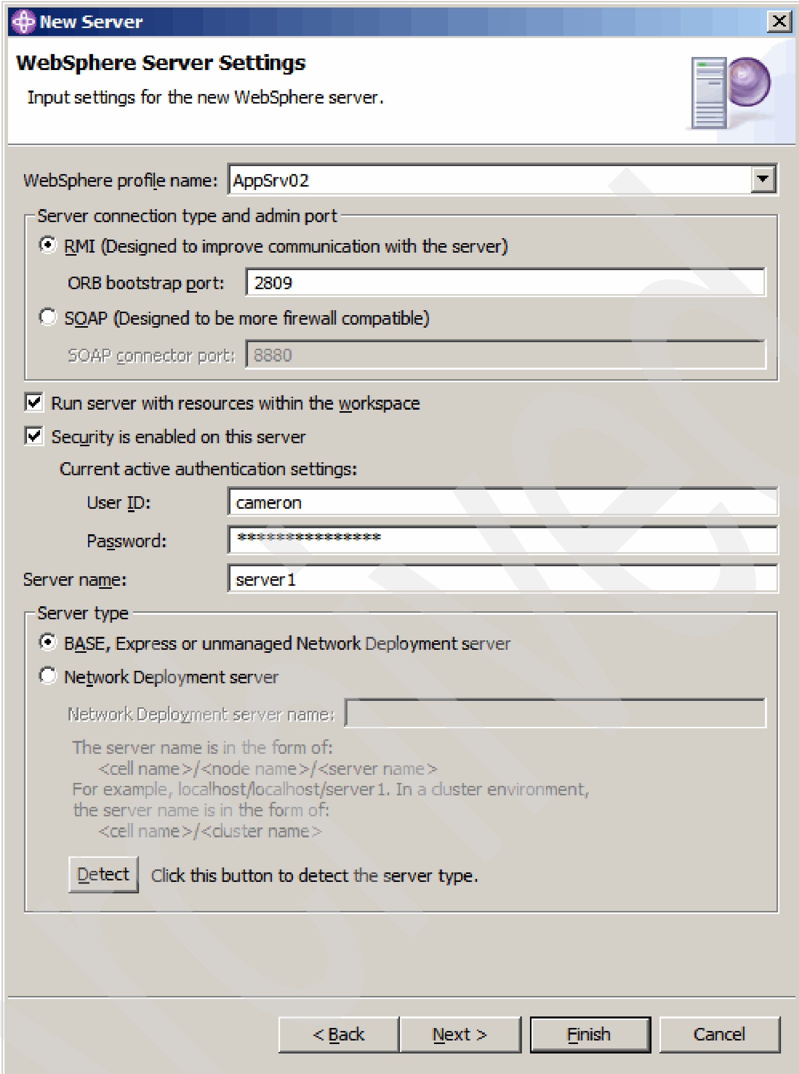


Figure 10-28 WebSphere Server Settings

5. Configure the WebSphere Profile that you will use for testing.
   a. Select the WebSphere profile that you want to use in the WebSphere profile name drop down.
   b. If security is enabled, ensure that the Security is enabled on this server check box is checked and configure authentication as follows:
      i. In the User ID field, enter the username entered earlier.

ii. In the Password field, enter the password entered earlier.

c. Click **Next** to continue.

6. The Add and Remove project dialog will now display as shown in Figure 10-29.

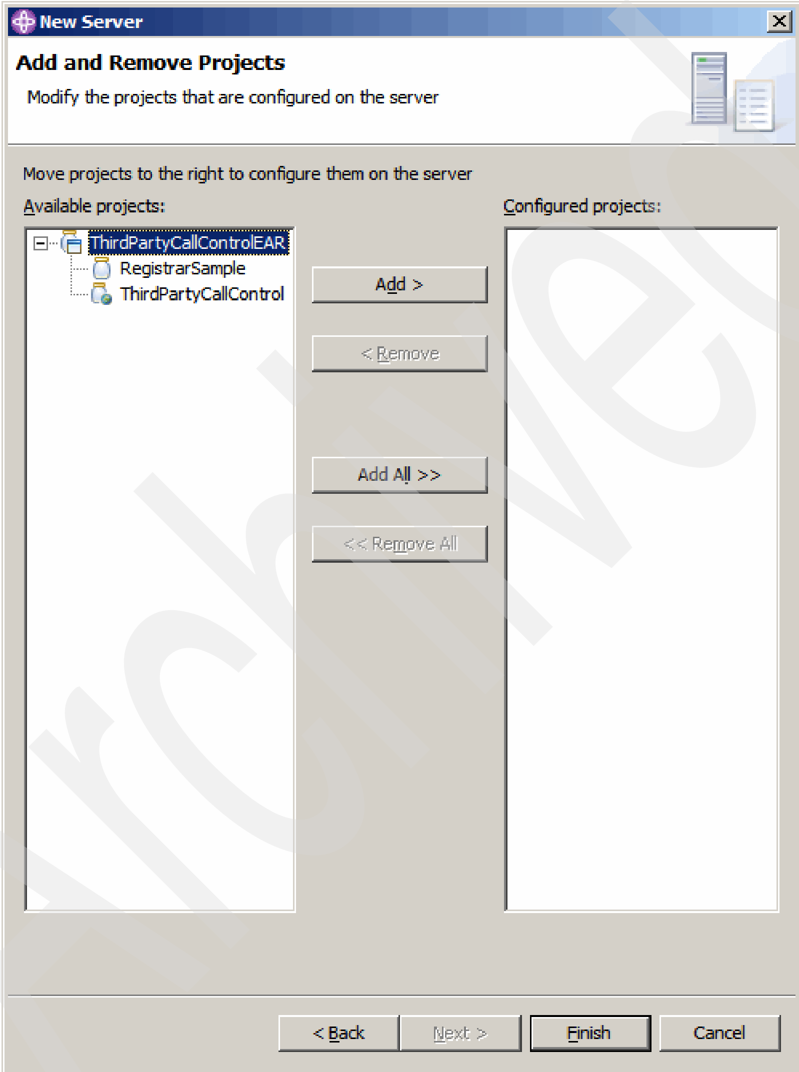


Figure 10-29 Add and Remove Projects

7. Add the EAR file to the list of Configured projects and click **Add**.
8. Complete the configuration and click **Finish**.

9. A new Server has now been added to the list of Servers as shown in Figure 10-30.

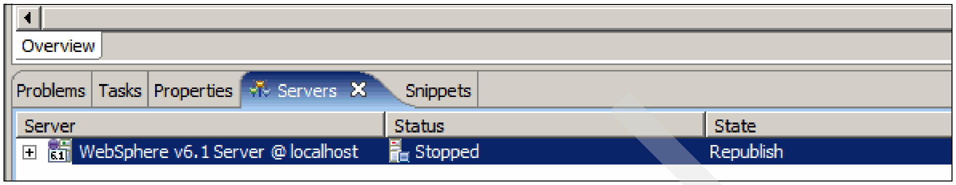


Figure 10-30 WebSphere v6.1 Server @ localhost

10. Double-click the server that you configured. A window similar to Figure 10-31 on page 285 will be displayed.

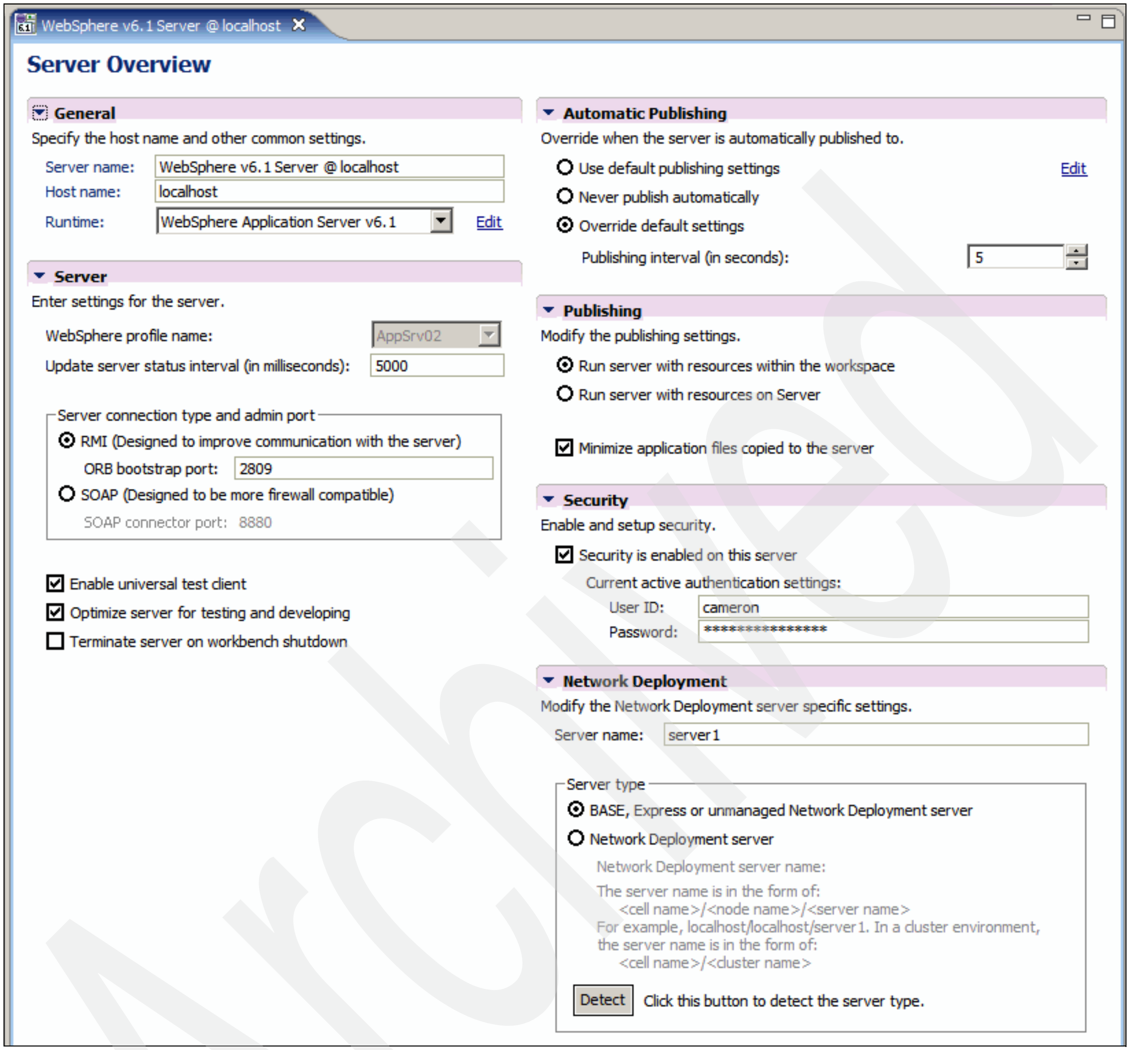


Figure 10-31 Server Overview

> **Note:** If you want to make further changes to the Server Configuration, they can be performed using this panel. For example, if you want to change the frequency of Automatic Publishing from 5 seconds to some other value, then this can be changed on this page.

10.4.5 Testing the Third Party Call Control application

To test this application, we will use the same User Agents configuration as the previous example. The two User Agents will operate with the WebSphere Application Server running on the same machine.

Start the Server

1. Right-click the server listed in the Server Panel.

   A menu as shown in Figure 10-32 will be displayed.

2. Select Start to start WebSphere Application Server.

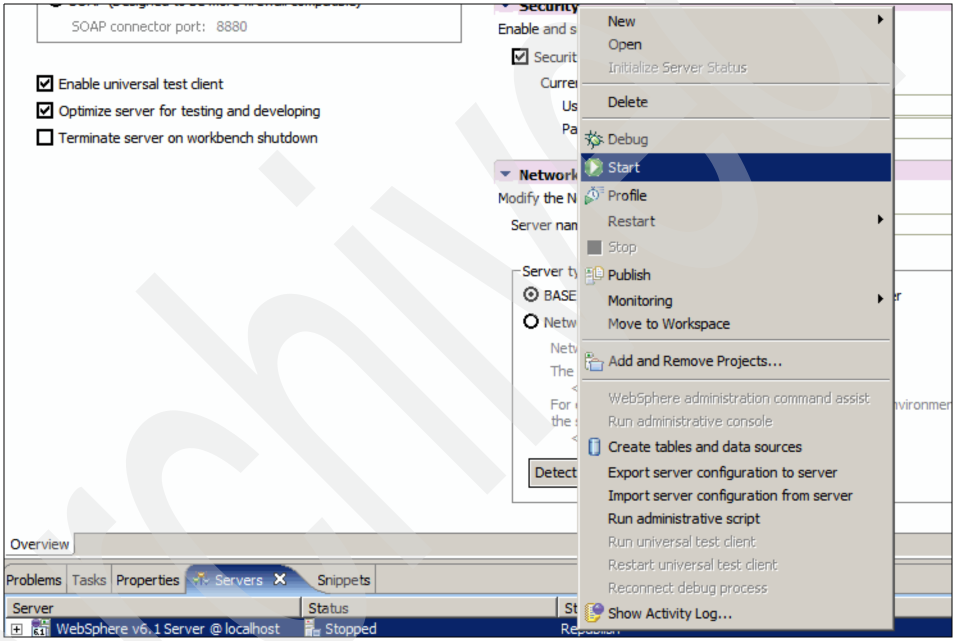


Figure 10-32 Select Start from the Server Control Menu

3. A Console tab will now be opened, and status information of the server start-up will be displayed.

Start the User Agents

1. Start sipXphone.
2. Start X-Lite.

Both clients should register successfully as they did in the previous sample.

Note: See 10.3.3, “Testing the Registrar and proxy application” on page 253 for steps on how to start the sipXphone and X-Lite softphones.

Test the Application

1. Within the Project Explorer Panel.
   a. Expand the ThirdPartyCallControl project → Servlets.
   b. Select CallControl.
2. Right-click CallControl.
3. Launch a browser window containing the Third Party Call Control Application. Select **Run As** → **Run on Server**.
4. The Define a New Server panel will be displayed as shown in Figure 10-33 on page 288.

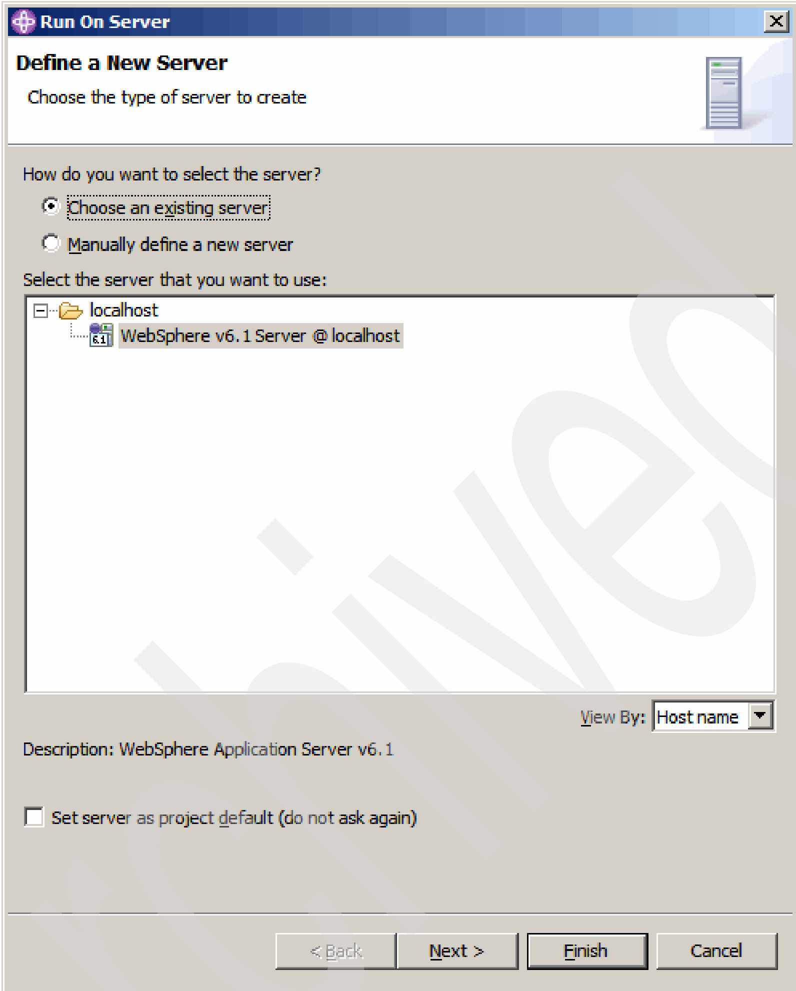


Figure 10-33 Run On Server

5. Accept the default selection of the existing server.
6. Click **Finish** to launch a test browser.
7. The test browser will launch as shown in Figure 10-34 on page 289.

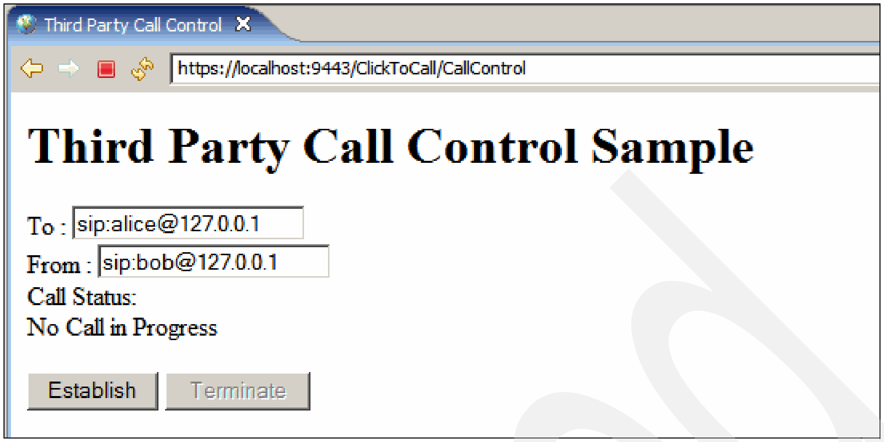


Figure 10-34 Third Party Call Control Sample

8. The SIP URIs to establish a call To and From have been pre-filled with sip:alice@127.0.0.1 and sip:bob@127.0.0.1. Click **Establish** to setup the call.
9. Alice's phone will now start ringing. Click the Off Hook button in X-Lite to accept the call.
10. Bob's phone will now start ringing. Click the Off Hook button in sipXphone to accept the call.
11. As the session continues, the Third Party Call Control Sample status page changes to reflect the status of the call.
12. In sipXphone, click Disconnect to terminate the call.

You have now seen the Third Party Call Control application in use.

10.4.6 Debug and trace the application

You will use the integrated debugging support in the AST to set breakpoints in the execution of the application, step through the execution of the code and inspect the value of variables.

1. Switch to the Java perspective.
2. In the Project Explorer, expand the **ThirdPartyCallControl** folder.
3. Expand **src** → **com.ibm.itso.sg247255.sample2** → **CallController.java**.
4. Select the **doSuccessResponse** method to navigate to this method.
5. Set a breakpoint by clicking twice in the gray bar to the left of the source code. A breakpoint marker will display as shown in Figure 10-35 on page 290.

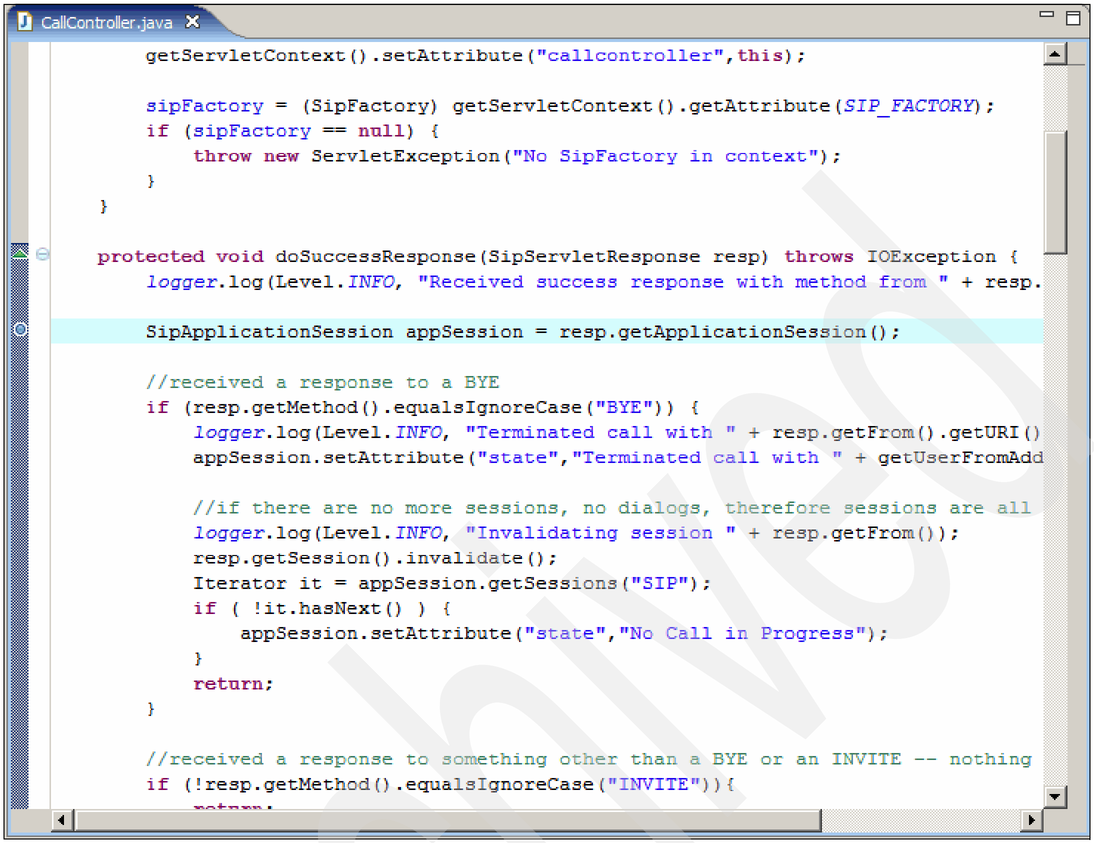


Figure 10-35 Setting a Breakpoint in CallController

> **Note:** In order for the debugger to be active, the server must be started in debug mode. If the Application Server is already running, stop it by clicking the Stop the Server button on the left-hand side of the Server panel.
>
> If sipXphone and X-Lite are already running, you need to stop these as well.

6. Right-click the WebSphere V6.1 @ localhost server. Select the option debug to start the server in debug mode.
7. The server will now commence starting up in debug mode. Once the server has started, you will be prompted to switch to the Debug perspective as shown in Figure 10-36 on page 291.

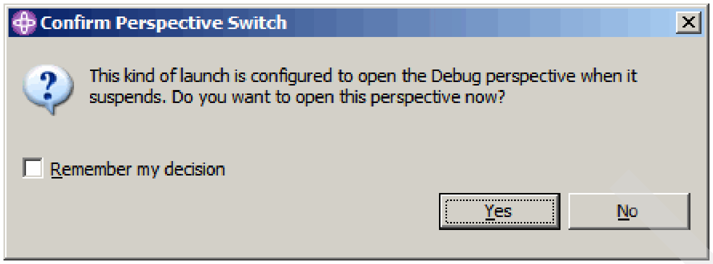


Figure 10-36 Confirm Perspective Switch

8. Click **Yes** to switch to the Debug perspective. The Debug perspective will be displayed similar to Figure 10-37 on page 292.

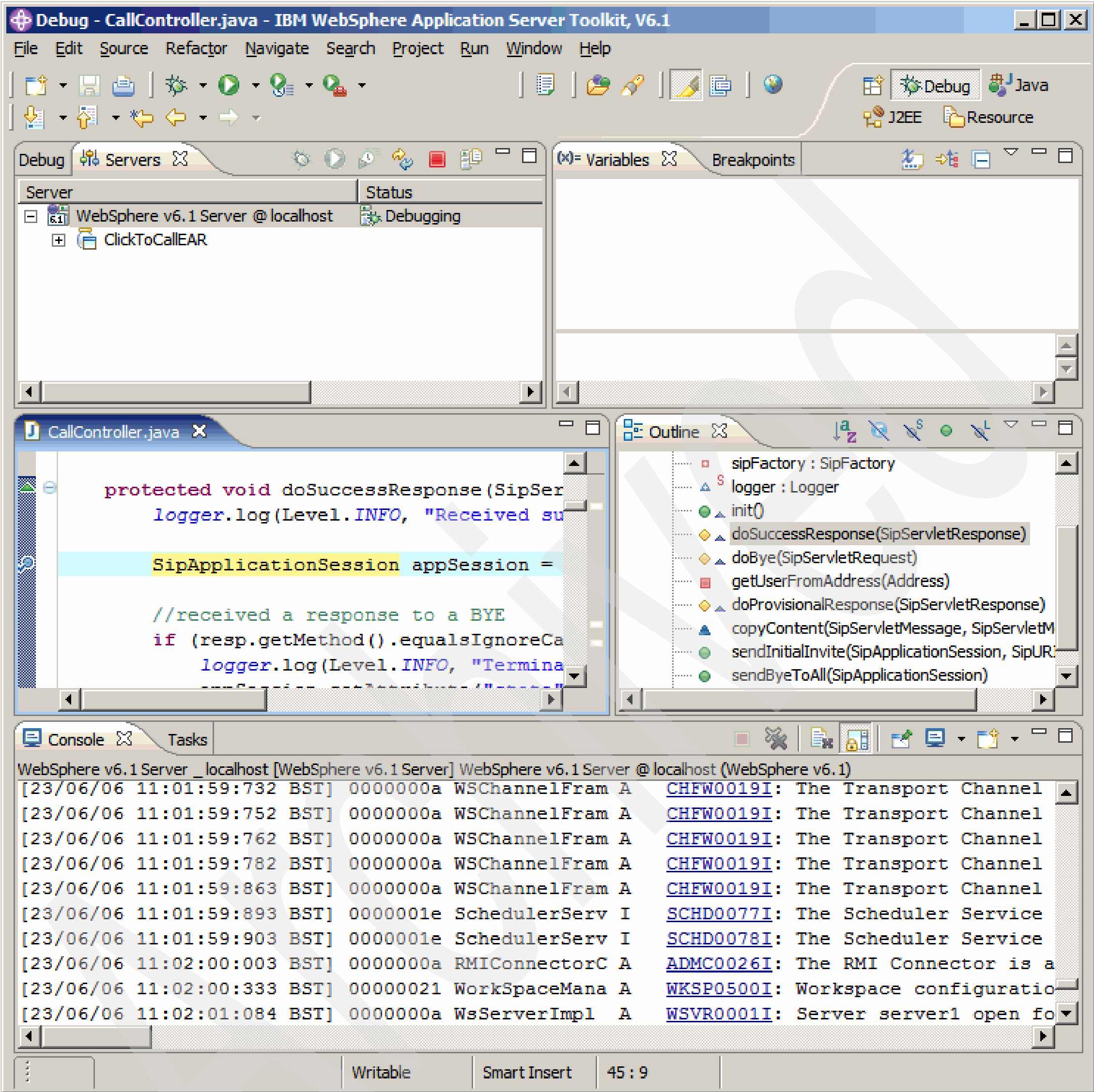


Figure 10-37 Debug Perspective

9. Start sipXphone and X-Lite.
10. Switch to the Java Perspective and launch the Click to Call Sample application using the same steps used in “Test the Application” on page 287.

11. Once the call has been answered in X-Lite, the focus in the Application Server Toolkit will highlight the thread of execution which has been suspended as shown in Figure 10-38.

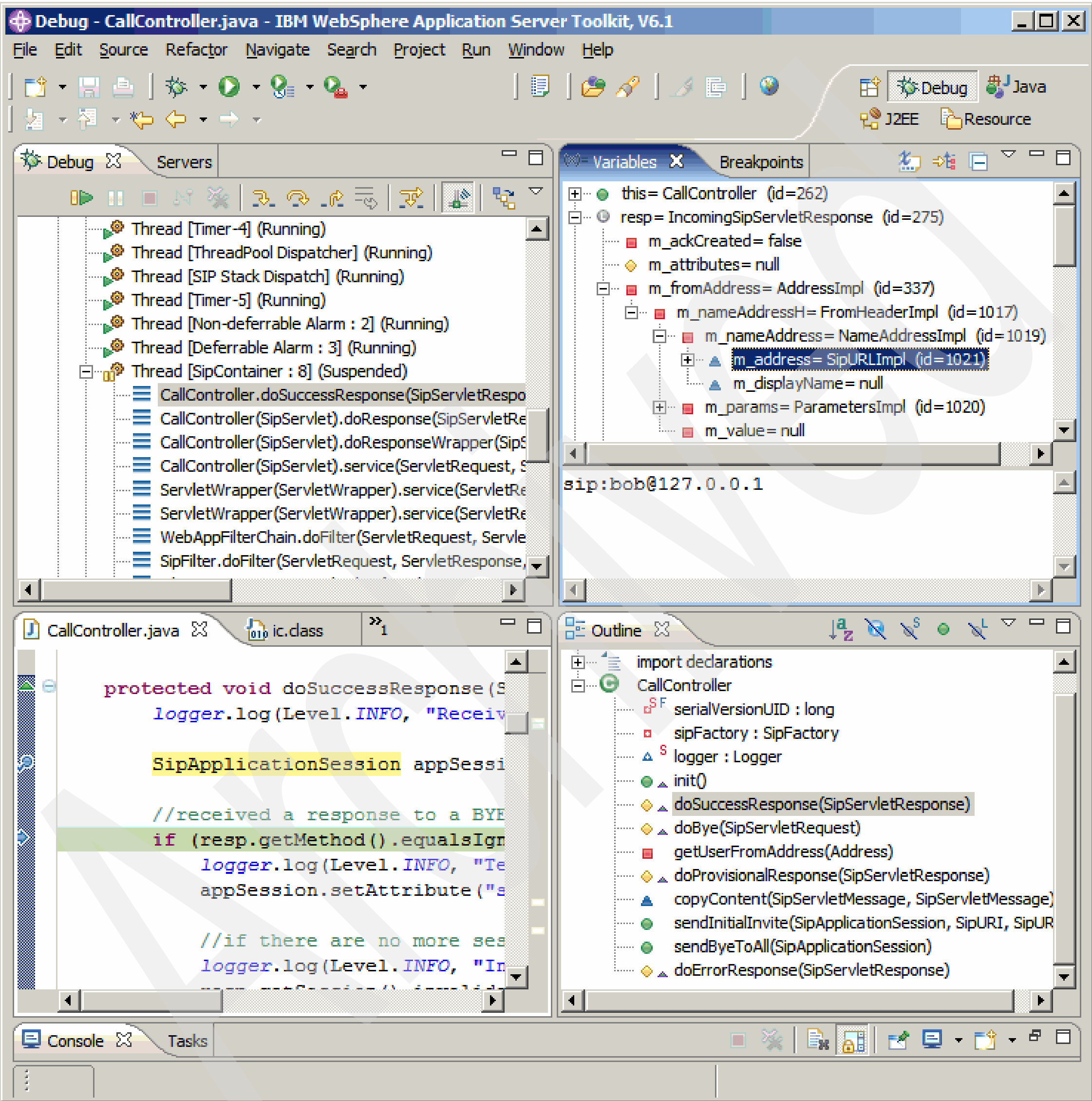


Figure 10-38 CallController Debug in Progress

12. The debug tools of Resume, Step Into, Step Over, Step Return are available in the top row of the Debug panel.

13. Variables may be inspected using the Variables panel. As shown in Figure 10-38 on page 293, the IncomingSipServletResponse may be inspected, and the contents of this object inspected further. For example, the contents of the From Header are shown above as being sip:bob@127.0.0.1.

14. Press Resume to continue execution of the application.

You have now seen how to set a breakpoint and debug a SipServlet executing with the test environment.

Tracing the SIP Container

In order to see the flow of SIP messages through WebSphere Application Server, tracing of the SIP container may be performed. This may be useful particularly when inspecting the operation of a converged application, or where multiple resources are invoked.

1. Switch to the J2EE Perspective.
2. Right-click the WebSphere v6.1 @ localhost server.
3. Select Run Administrative Console.
4. Log onto the Administration Console by entering your username and password.
5. Expand **Troubleshooting**.
6. Select **Logs and Trace**.
7. Select **server1** → **Diagnostic Trace** → **Runtime** Underneath Additional Properties.
8. Select Change Log Detail levels.
9. Change the log trace string to `*=info: com.ibm.ws.sip.*=all`. The console will appear similar to that shown in Figure 10-39 on page 295.

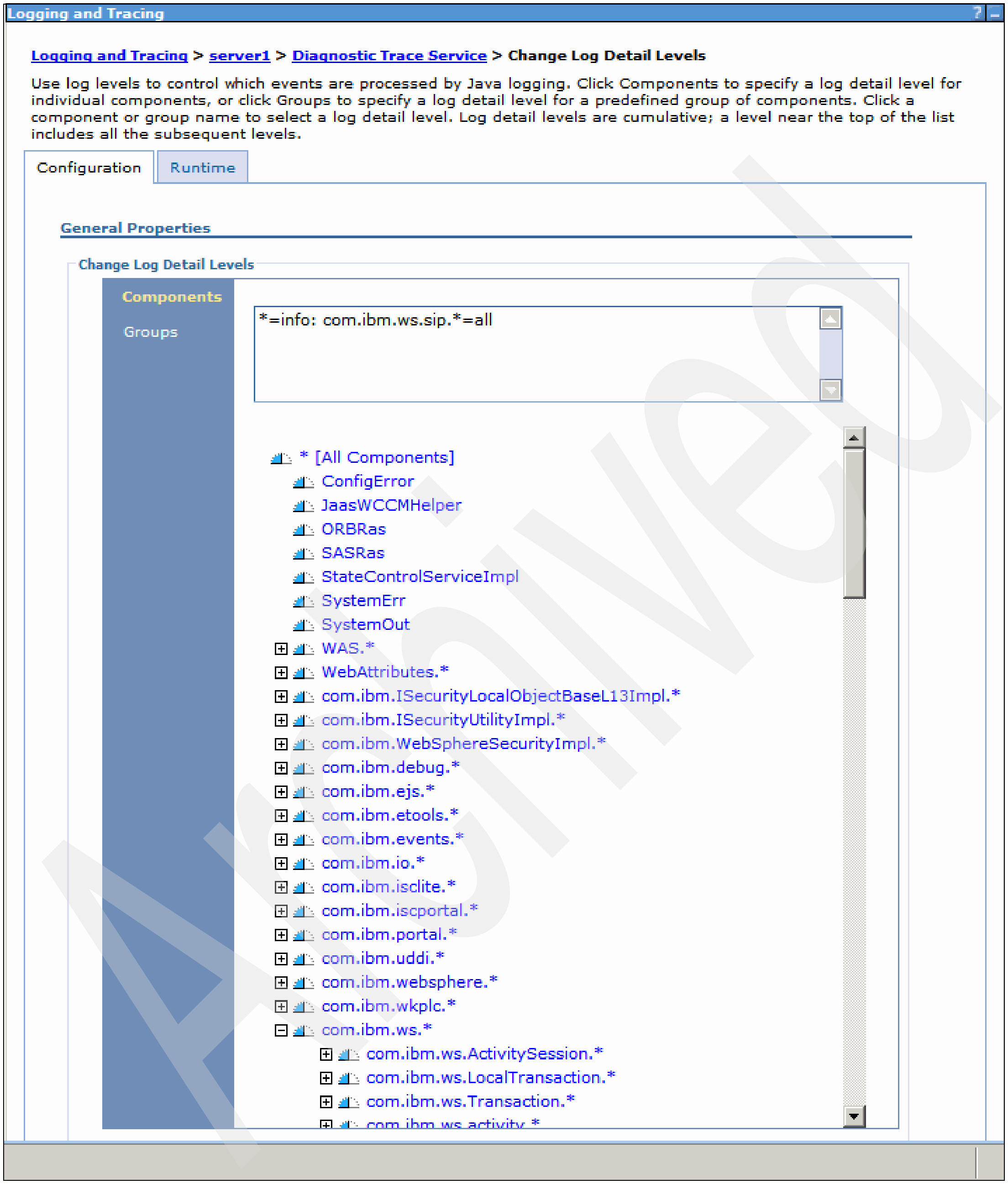


Figure 10-39 Configuring SIP Tracing

10.Click **OK**.

11. Click **Save** to save the changes to the master configuration.

12. The runtime will now output trace information matching the trace specification you entered.

13. Load the trace.log file. This will be located in the Application Server's log directory. An example of the output created is shown in Example 10-16.

Example 10-16 Example trace output

```
[23/06/06 11:38:17:043 BST] 00000090 IncomingMessa 3
IncomingMessageEvent  IncomingMessageEvent [91] queued
[23/06/06 11:38:17:043 BST] 00000090 SIPUdpConnect >
SIPUdpConnection$SIPReadHandler: read: entry: id=888943868 Entry
[23/06/06 11:38:17:043 BST] 00000090 SIPUdpConnect >
SIPUdpConnection$SIPReadHandler: validateReadRequest: entry:
id=888943868 Entry
[23/06/06 11:38:17:043 BST] 00000090 SIPUdpConnect <
SIPUdpConnection$SIPReadHandler: validateReadRequest: exit:
id=888943868 Exit
[23/06/06 11:38:17:043 BST] 00000090 SIPMessageFac >
SIPMessageFactory:getObject: entry: id=77857956 Entry
[23/06/06 11:38:17:043 BST] 00000090 SIPMessageFac 3
SIPMessageFactory:object removed from object pool
[23/06/06 11:38:17:043 BST] 00000090 SIPMessageFac <
SIPMessageFactory:getObject: exit: id=77857956 Exit
[23/06/06 11:38:17:043 BST] 00000090 SIPUdpConnect <
SIPUdpConnection$SIPReadHandler: read: exit: id=888943868 Exit
[23/06/06 11:38:17:043 BST] 00000090 SIPUdpConnect <
SIPUdpConnection$SIPReadHandler: complete: exit: id=888943868 Exit
[23/06/06 11:38:17:043 BST] 0000002a IncomingMessa 3
IncomingMessageEvent  IncomingMessageEvent [91] dispatched from
[127.0.0.1:3834/UDP]
SIP/2.0 200 OK
Via: SIP/2.0/UDP
NOMAD.uk.ibm.com:5060;branch=z9hG4bK936417551780891;received=127.0.0.1;
ibmsid=local.1151056881297_7_7
Via: SIP/2.0/UDP
NOMAD.uk.ibm.com:5060;ibmsid=local.1151056881297_6_6;branch=z9hG4bK6385
92172166077
Contact: <sip:alice@127.0.0.1:3834;rinstance=faacca596eda9e86>
To: <sip:alice@127.0.0.1>;tag=683a2071
From: <sip:bob@127.0.0.1>;tag=9561759554153756_local.1151056881297_6_6
Call-ID: 6957398777660333@NOMAD.uk.ibm.com
CSeq: 2 INVITE
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE,
SUBSCRIBE, INFO
```

```
Content-Type: application/sdp
User-Agent: X-Lite release 1002tx stamp 29712
Content-Length: 429

v=0
o=- 3 2 IN IP4 9.146.11.27
s=<CounterPath eyeBeam 1.5>
c=IN IP4 9.146.11.27
t=0 0
m=audio 4140 RTP/AVP 107 119 6 0 98 8 3 5 101
a=alt:1 2 : KxLZOacl W1gIOtTw 9.146.11.27 4140
a=alt:2 1 : +RWFpHC/ UVtoXZDU 192.168.37.1 4140
a=fmtp:101 0-15
a=rtpmap:107 BV32/16000
a=rtpmap:119 BV32-FEC/16000
a=rtpmap:98 iLBC/8000
a=rtpmap:101 telephone-event/8000
a=sendrecv
a=x-rtp-session-id:9FE72A42BF9142D38C141690A9C0AECA
```

You have now seen how to trace the application and inspect the flow of events through the SIP container.

Part 4

Developing IMS applications

Part 4 provides guidelines for developing applications that use the IBM WebSphere IP Multimedia Subsystem Connector, the IBM WebSphere Presence Server and the IBM WebSphere Telecom Web Services Server. The working examples demonstrate how to create composite services.

11

Designing IMS services

This chapter provides an overview of composite services and introduces the design process with relevant guidelines for creating services that use the IBM IMS solution products. It also provides the description of the design for *FindHelp*, the sample IMS service implemented in this redbook.

This chapter contains the following:

- Overview of IMS composite services
- Designing composite services
- Sample application design

11.1 Overview of IMS composite services

IP-based networks and services are enabling the migration to full convergence where the use of standard based framework allow for the deployment of integrated Web based applications that leverage the underlying telecommunications networks.

In this section we introduce IBM enablement solution for composite services and identify the benefits associated with creating services in this manner. We also provide design guidelines for creating IMS composite services.

IMS composite services are created by combining two or more service building blocks.

- IMS composite service

  Composite service is defined as end user service created utilizing reusable service building blocks.

- Service building block

  Service building blocks are components that exposes well defined interfaces that can be consumed by calling entities. Service building blocks can be separated into two categories:

  - Enablers

    These are normally Web service entities that are not sold to end users. Instead they are combined to provide services for example location service and map service. Note that an enabler can be a SIP enabled application that is exposed as a Web service such as Presence Server.

  - Foundation services

    Foundation services are SIP based entities that can be combined to create new services. They are available as stand-alone services. Example foundations include conferencing and multiplayer gaming.

11.1.1 Composite services architecture

The IBM IMS composite service architecture supports two categories of service composition:

- Service Orchestration

  This involves the composition of foundation services utilizing either the CSCF or Service Capability Interaction Manager (SCIM) within the IMS architecture. The CSCF is used to compose services if the foundation services act as SIP Proxies where the service acts as a user agent server and consumes the last SIP message. A SCIM is required if multiple foundation services act as user agent servers. The original SIP message is sent to all services and the responses are correlated into a single response. Currently the SCIM is loosely defined within the 3GPP specifications.

- Service Choreography

  This involves the composition of enablers utilizing a Business Process Execution Language (BPEL) engine such as WebSphere Process Server.

Figure 11-1 is an illustration of the overview of the IBM IMS service composition.

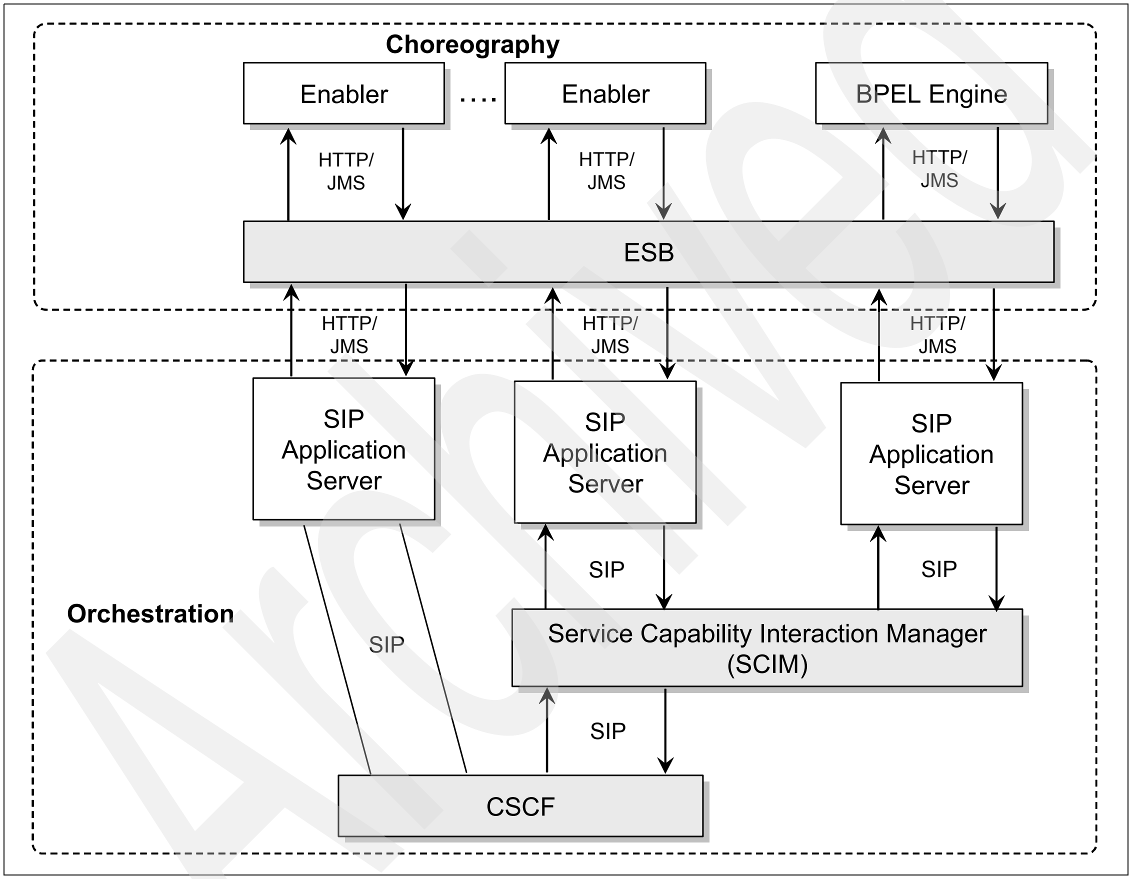


Figure 11-1 IBM IMS service composition

11.2 Composite services choreography

As mentioned above, service choreography is utilized when composing enablers that are implemented as Web Services. The IBM IMS service composition architecture includes the following components which support service choreography:

- SIP Application Server

  The SIP application server receives SIP based message and determines if service choreography is required. The message is parsed and passed to an endpoint exposed on the enterprise service bus.

- Enterprise Service Bus

  The Enterprise Service Bus (ESB) provides the middleware to convert protocol and format of inbound requests. It routes requests to the BPEL Engine, and additionally will complete the conversion and routing for exposed enablers such as Location Server and Map Service.

- BPEL Engine

  The BPEL engine provides the service choreography engine. This usually involves the composition of two or more Web services.

Figure 11-2 shows the architecture of a Route Finder application that utilizes service choreography to retrieve the users current location followed by a map and route request for the user.

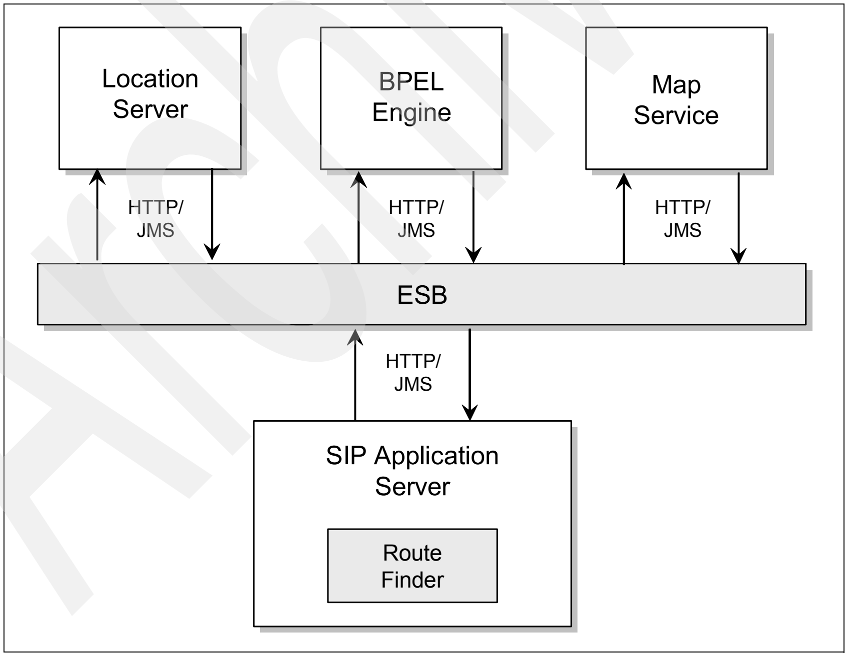


Figure 11-2 Service Choreography

11.2.1 Composite services orchestration

Service orchestration composites foundation services. The SIP composite service shown in Figure 11-3 is based on foundation services acting as user agent servers and therefore require composition by a SCIM. Requests are forwarded by the CSCF to the SCIM, which then sends the requests to the multiplayer gaming and conferencing services. The responses are sent through the SCIM which is responsible for collating these into a single response that is sent to the client. The client is unaware of the services orchestration.

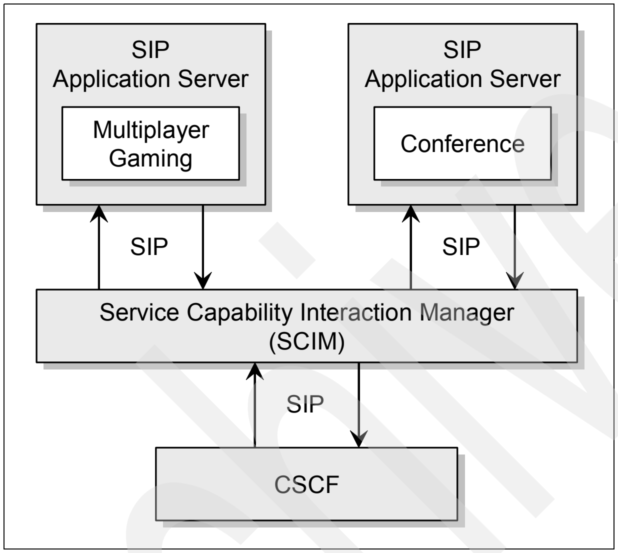


Figure 11-3 Service Orchestration

11.3 Designing composite services

This section introduces the design process and provides design guidelines that are relevant for the IBM IMS solution products. It considers when it is appropriate to utilize SIP Web Services, how and when to integrate BPEL into a service and the options available for an Enterprise Service Bus.

11.3.1 Design process

Designing a composite service can be separated into the following steps:

- Analysis

Analyze the end user service and identify the service building blocks required. This process is similar in practice to service identification within SOA. Here are a number of resources that provide guidance:

– *Patterns: Service-oriented Architecture and Web Services*, SG24-6303

  http://www.redbooks.ibm.com/abstracts/sg246303.html

– *Service-oriented modeling and architecture*

  http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/

– *Elements of Service-Oriented Analysis and Design*

  http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/

- Flow consideration

  Consider if service orchestration and/or choreography is appropriate within this service.

- Component selection

  Search for existing service building blocks for the appropriate services. It is important to reused service building blocks were possible to get the most benefit out of service composition. Therefore it is critical that you maintain service building block repository that is searchable to assist reuse. Here is a list of suggested reading materials regarding reusable repository that may be considered:

  – *Building SOA applications with reusable assets: Reusable assets, recipes, and patterns*

    http://www-128.ibm.com/developerworks/webservices/library/ws-soa-reuse1/

  – *Reusable Asset Specification Repository for Workgroups*

    http://www.alphaworks.ibm.com/tech/rasr4w

- Solution design

  This is the final step in the design process. The use of model driven design will enable you to create a comprehensive design for your solution. IBM provides Rational Unified Process® and Unified modelling language (UML) based tools that has successfully been applied in several industries including telecommunications. Consult the following resources for more information:

  – *UML basics: An introduction to the Unified Modeling Language*

    http://www-128.ibm.com/developerworks/rational/library/769.html

– *Rational UML Profile for business modeling*

http://www-128.ibm.com/developerworks/rational/library/5167.html

– *Introducing IBM Rational Software Architect*

http://www-128.ibm.com/developerworks/rational/library/05/524_rsa/

– *Patterns: Model-Driven Development Using IBM Rational Software Architect*, SG24-7105

11.3.2 SIP Servlets as Web Services

WebSphere Application Server 6.1 converged SIP and HTTP container provides a simple mechanism to exposing SIP Servlets as Web Services, however care must be taken to ensure that you do not over engineer solutions with unnecessary encapsulations. We will discuss a number of scenarios and suggests best practices from SIP Web Services stand point.

Invoking BPEL from SIP Servlets

The SIP Servlets are invoked from User Agent clients and during the execution of servlets, BPEL flows are invoked. Either during or at the completion of a BPEL flow, additional SIP communication with the SIP User agent client is required. Here are two approaches for realizing the SIP communication with the SIP User agent client:

- Web Service interface

  The BPEL could invoke a SIP Servlet through a Web Service interface, and communicate either on a different dialog from the original, or ensure that the same dialog is utilized. Figure 11-4 on page 308 shows an illustration of this option.

  Additional logic may be required in the User Agent client to correlate the two dialogs, if that is the case, then the use of this option is discouraged.

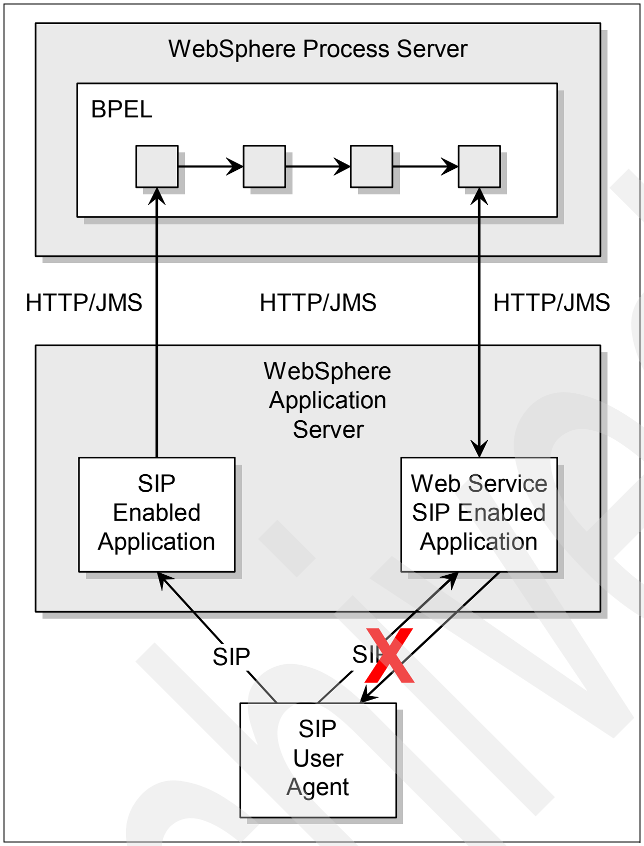


Figure 11-4 Unrequired SIP Servlet exposed as a Web service

- SIP Servlet pass-through

  In general, the recommendation would be for the BPEL to return to the calling SIP Servlet relevant information such that it can communicate with the User Agent client on behalf of the BPEL and if required the SIP Servlet can pass data back to the BPEL flow. An illustration of this is shown in Figure 11-5 on page 309.

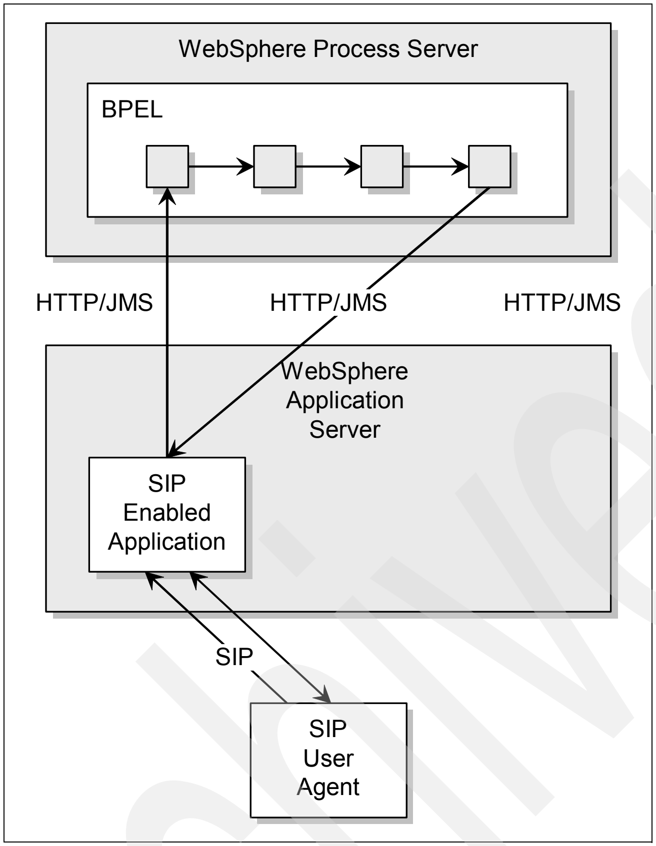


Figure 11-5 Preferred architecture for SIP to BPEL communication

> **Note:** As with all guidelines there are exceptions and if the BPEL flow was to include Human Tasks or other long term processing, and the average processing time is over 10 seconds (use only as a rough guideline), then it is reasonable to suggest that the original SIP dialog may be closed and a new dialog establish when the BPEL process is ready for SIP communication to occur.

BPEL communication with SIP enabled application

In this scenario, the BPEL process is started via another SIP Servlet or an external mechanism such as a Web Service. The SIP enabled application required by the BPEL flow is completely independent to the calling entity and does not require any direct interaction with any existing SIP dialogs. A real life example would be a BPEL flow that requires interaction with a SIP enabled Presence Server to retrieve status information. In this instance it is suggested that the SIP Servlet be encapsulated as a Web Service and invoked directly by the BPEL. An illustration of this architecture is shown in Figure 11-6.

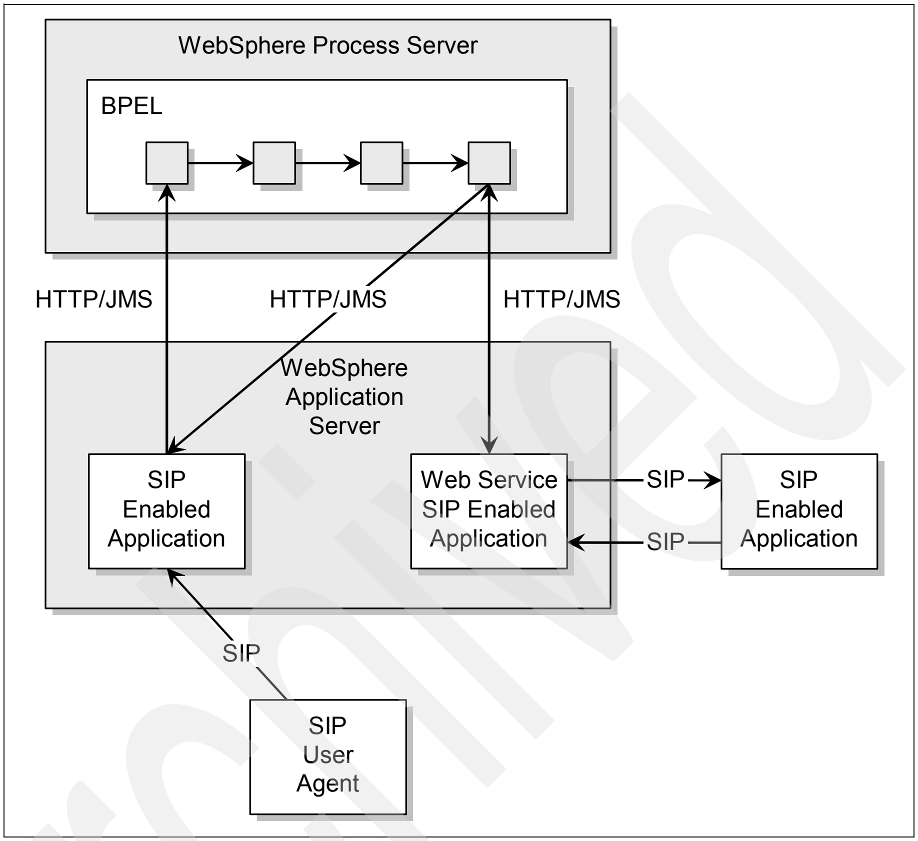


Figure 11-6 BPEL communication with a SIP enabled application

Note: If the BPEL was invoked by a SIP Servlet then the Web Services encapsulated SIP Servlet called by the BPEL should have no direct implications on the existing SIP sessions between the server and User Agent client.

The above scenarios focus on BPEL invoking SIP Web Services, however it is common to have SIP Web Services exposed on an Enterprise Service Bus for any Web Service consumer, in that situation it is unlikely that a SIP dialog would already exist, and simply exposing the SIP Servlet as a Web service is recommended.

11.3.3 Deciding when to use BPEL

BPEL provides a powerful mechanism for creating dynamic flows that can span people, systems, applications, tasks, rules, and the interactions among them. To realize the full benefits of BPEL is important to understand when to use BPEL and when other options many be more appropriate. The following list provide considerations you should take into account when developing services to determine if BPEL is appropriate:

- Latency

  BPEL is a powerful tool, however there are some latency issues that need to be considered before recommending its use in a service. If the expectation is for a sub-second end to end response then it is unlikely that BPEL would meet these requirements. This is due to the fact that other components within the architecture consume several hundred milliseconds, leaving BPEL with little time for processing. Therefore the overhead related to BPEL would not be suitable for such low latency applications. If the end to end response time of the application is in the order of several seconds then BPEL fits extremely well into this solution and can provide substantial benefits. In some countries there are legal requirements regarding the time taken to establish a call, however in other situations such as text messaging it is common to wait several seconds for the service to complete. The latency of BPEL has to be evaluated on a service by service basis depending on the expectation.

- Number of Web Services involved

  Due to the exciting capabilities of BPEL there is the misconception that all IMS services must utilize BPEL specially if Web Services is required. If there are multiple Web Services that need orchestration then BPEL is the best option to accomplish this, however if a single Web service needs to be invoked for backend integration then it may be more straightforward to call this Web service directly from the SIP Servlet code.

- Future modifications

  If the logic for a given service is likely to be modified regularly in the future for further enhancements, in particular the Web Service orchestration logic. It is sensible to consider using BPEL even if there is only one Web Service when the service is initially created. This is due to WebSphere Integration Developer and visual tools available for modifying BPELs.

- Human Tasks

  WebSphere Process Server provides extensive support for human tasks. If integration of a SIP initiated process requires human tasks it may be appropriate to consider using WebSphere Process Server within the solution.

11.3.4 Choosing ESB Software

IBM currently offers a number of Enterprise Service Bus (ESB) products that can be integrated into an IMS Service Plane solutions. The following are some ESB products and the recommendations regarding what best fits into the IMS Service Plane architecture:

- WebSphere Enterprise Service Bus (WESB)

  WESB is designed to provide the core functionality of an ESB for a predominantly Web Services based environment. It is built on WebSphere Application Server, which provides the foundation for the transport layer. WebSphere Enterprise Service Bus adds a mediation layer based on Service Component Architecture (SCA) programming model on top of this foundation to provide intelligent connectivity. If you have a lot of Web Services in your environment, WebSphere Enterprise Service Bus is likely to be the product to use.

- WebSphere Message Broker

  WebSphere Message Broker provides an advanced ESB with advanced integration capabilities such as universal connectivity and any-to-any transformation for data-centric deployments. It can handle services integration as well as integration with non-services applications. WebSphere MQ provides the transport backbone for messaging applications. Typically, customers who need high throughput product in a message-centric environment should use WebSphere Message Broker.

For further reading, consult these resources:

- *Getting Started with WebSphere Enterprise Service Bus V6,* SG24-7212
- *Enabling SOA Using WebSphere Messaging*, SG24-7163

11.4 Sample application design

In this section we introduce an IMS sample application called *FindHelp*. We present the specification from a business perspective, an end-user perspective as well as from an IT perspective and discuss the design of the sample application.

11.4.1 Objectives of the sample application

The FindHelp application design covers all major elements of an IMS service architecture. The design will demonstrate the following:

- How SIP Servlets access IMS enablers such as the Presence service and the Group List service.
- How SIP Servlet interact and integrate with process choreography.
- How to choreograph service enablers such as Location, Charging and Call Control services using BPEL processes.

11.4.2 The business scenario

The FindHelp sample service involves three parties:

- A subscriber

  The subscriber is entitled to use the FindHelp service, because he has subscribed to it.
- Service provider

  A telecommunications service provider, who offers the FindHelp service.
- Lock service company

  A company that provides lock services and has setup agreement with the service provider to be included in the FindHelp service.

Here is the description of the scenario:

- A lock service company offer locksmith services across a certain region. Their staff are highly skilled technicians, can open car, home or any other door for customers who have either misplaced, stolen or broken keys. As a way to reach out to potential clients ACME has set up an agreement with the regional service provider to be listed in the service provider's FindHelp service.
- Using an application that is provided by the service provider, the ACME administrator defines a *Lock Out Services* topic and assigns his best technicians to this topic.
- ACME provides all its technicians with SIP enabled mobile devices, so that they can publish their current presence status.
- Bob, who is a subscriber of the service provider, comes home late. While trying to open the door of his apartment, Bob breaks the door key.
- Using his mobile phone Bob starts the FindHelp service and selects *Lock Out Services.*
- The FindHelp service identifies Jack, who is member of the *LockOutServices* group and who is currently active and is actually just around the corner of Bob's apartment.
- The FindHelp service establishes a call between Bob and Jack.

- After trying to support Bob by the phone, Jack finally decides to drive to Bob's apartment to solve the problem.

11.4.3 The use case model

Figure 11-7 on page 315 shows the use case diagram of the FindHelp application.

The human actors that interact with the application are:

- Administrator

  The administrator works for the Lock service company. His role is to maintain the FindHelp topics and assign technicians to these topics.

- Technician

  The technician provides the requested service. He is contacted by the A-Party to provide help for a certain problem.

- A-Party

  The A-Party is a subscriber of a communication service provider. He uses the FindHelp application to get contact to a technician.

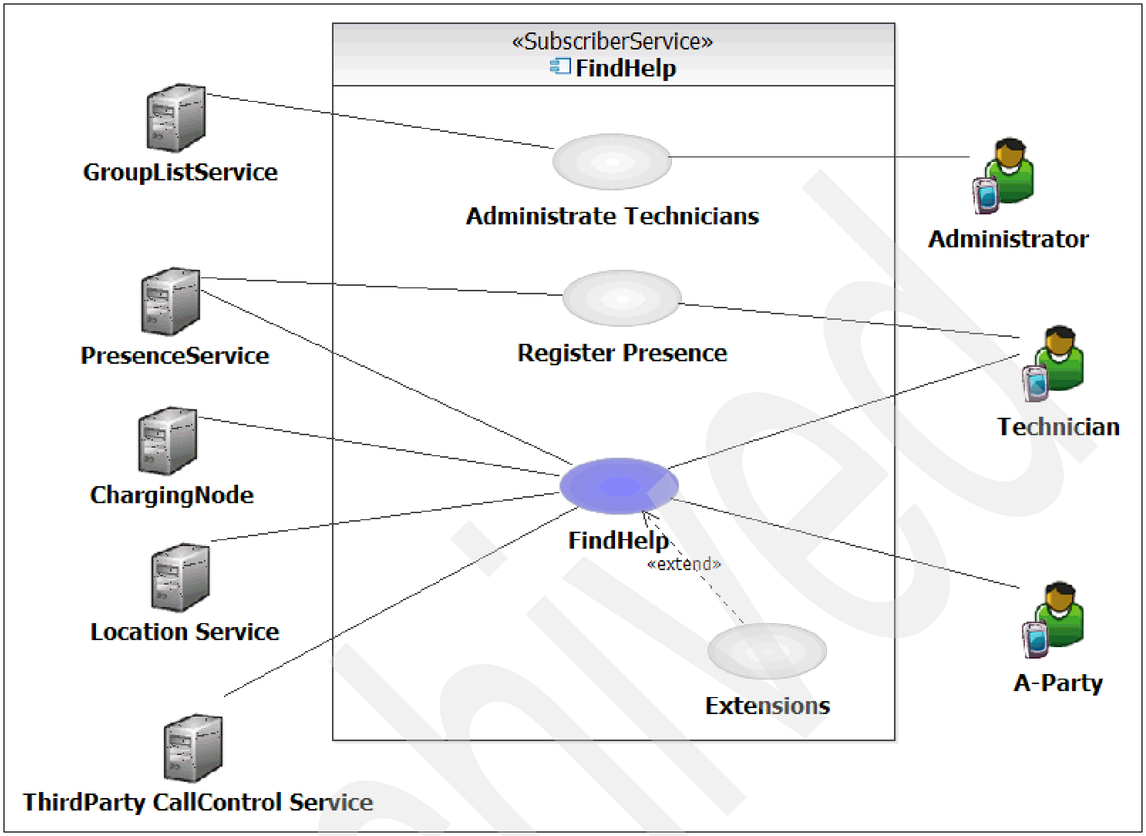


Figure 11-7 FindHelp Use Case Diagram

The FindHelp application also interacts with a set of enabling services:

- Group List service

  The Group List service provides means for administrating and querying groups and members.

- Presence service

  The Presence service provides presence information for a given address or group of addresses.

- Charging node

  The charging node collects charging relevant data.

- Location service

  The location service provides location and distance information of mobile subscribers.

- ThirdParty Call Control service

  The ThirdParty Call Control service provides means to establish or tear down calls between two subscribers.

The use cases

The main use cases of the FindHelp application are described as follows:

- Administer technicians

  This is the administration use case. The lock service company that wants to offer their services through the FindHelp service and as a result needs to maintain topics such as homecare, lockout service and pest control. It assigns technicians to these topics. It also maintains the profile of each technician.

  This use case is realized by the administration client that is part of the IBM WebSphere Group List Server Component. Refer to 13.3, “Executing the test scenarios” on page 466 for more detailed description.

- Register Presence

  The technician needs to publish his actual presence status. This use case is not implemented. However we simulate the functionality by using a SIPp script. Refer to 13.3, “Executing the test scenarios” on page 466 for a more detailed description.

- FindHelp

  This use case is described in this and subsequent chapters.

- Extensions

  The core FindHelp application may in future be extended by some additional features. Here are two examples:

  – Web access to the application

    Would enable A-Party to invoke the FindHelp application from a web-browser. This extension would introduce the use of the IBM WebSphere Telecom Web Services Server as B2B access gateway. It also would introduce the use of the converged container for SIP and J2EE servlets.

  – Provide directions to A-Party to technician

    The FindHelp application could interact with a mapping/routing service for directions from the technician to the A-Party. This information could be send to the technician.

11.4.4 The component model

A number of components need to collaborate together to provide the functionality required by the FindHelp use case. Figure 11-8 shows a static view of the component interaction model. In 11.4.5, “Component flow” on page 322 the dynamic view of the component interaction is provided.

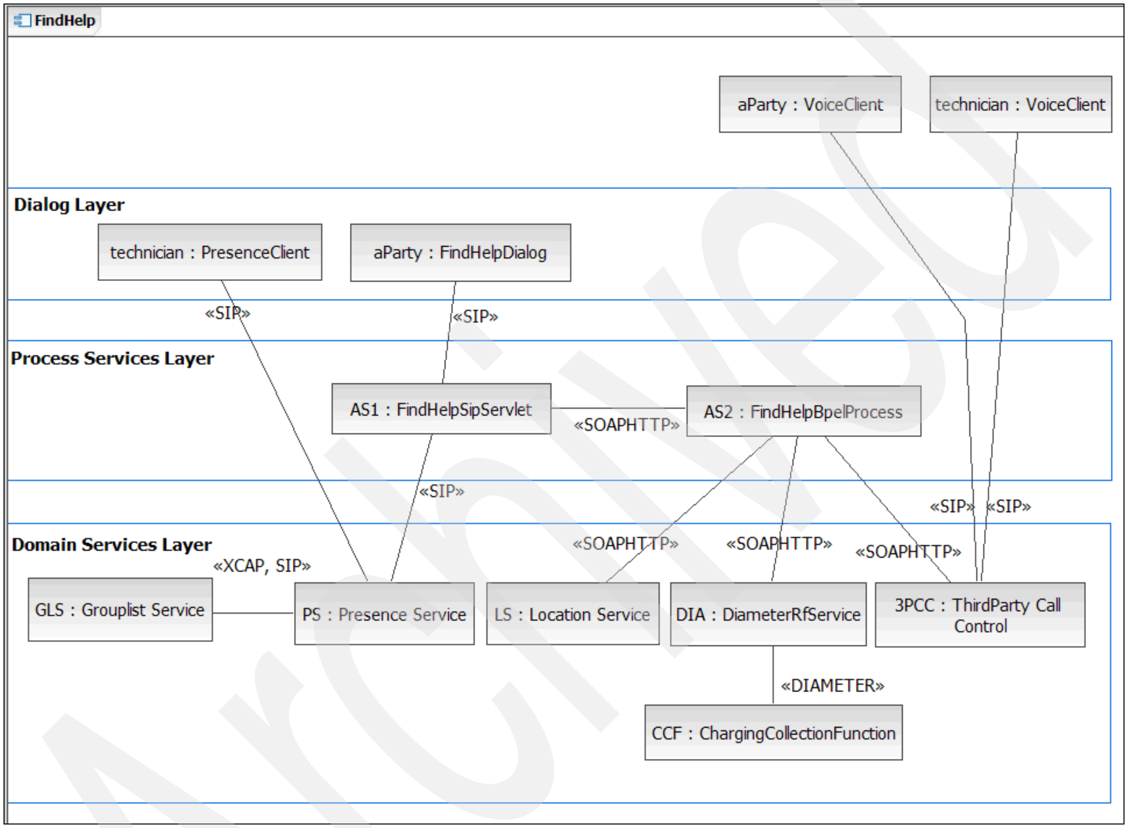


Figure 11-8 Component Diagram

The following outlines the key functionality and responsibilities of the components and the implementation:

- PresenceClient component
  - Responsibilities

    Notify the Presence Service about changes of the device’s presence status.

  - Collaborations

    Send SIP messages to the Presence Service.

  - Implementation

    For our sample we use a simulator (SIPp). The sequence of SIP messages is captured in a script.

    In a production environment this component would reside on, for example, a mobile device or represented by Web application.

  - Reference

    13.1, “Overview of the test environment” on page 440

- FindHelpDialog component
  - Responsibilities
    - Presents a list of topics
    - Implements the SIP client for the service
  - Collaborations

    Invokes the FindHelp service by sending a SIP message.

  - Implementation

    For our sample we use a simulator (SIPp). The sequence of SIP messages is captured in a script.

    In a production environment this component would reside on e.g. a mobile device or represented by Web application.

  - Reference

    13.1, “Overview of the test environment” on page 440

- VoiceClient component
  - Responsibilities

    Set up a voice communication channel.

  - Collaborations

    Receives SIP messages from the Third Party Call Control to setup a voice session.

  - Implementation

    For our sample we use SIP softphones like X-Lite and SipXphone.

    In a production environment this component would reside on either a SIP phone or a mobile device.

– Reference

13.2.4, “Device client setup” on page 459

- FindHelpSipServlet component
  - Responsibilities
    - Retrieve the closest subscriber
    - Establish a call between the A-party and the closest subscriber
    - Create a charging event
  - Collaborations
    - Receives SIP messages from the FindHelpDialog
    - Sends SIP messages to the FindHelpDialog
    - Start the BPEL process by invoking a Web service, which is provided by the FindHelpBpelProcess
    - Send SIP messages to the Presence service to retrieve the list of present subscribers
    - Receive SIP messages from the Presence service
  - Implementation

    Implemented as a siplet using the WebSphere Application Server Toolkit. The execution environment is the WebSphere Application Server V6.1.
  - Reference

    12.2, “SIP Servlet development” on page 342
- FindHelpBpelProcess component
  - Responsibilities
    - Resolve a topic to a list of subscribers, whose present status is active
    - Implement the FindHelp specific SIP state machine
    - Trigger the BPEL process
  - Collaborations
    - Provides a Web service, that is called by the FindHelpSipServlet
    - Invokes a Web service, that is provided by the Location Service, to get location and distance of subscribers
    - Invokes a Web service, that is provided by the Third Party Call Control, to establish a call
    - Invokes a Web service, that is provided by the DiameterRfService, to create a charging event

  - Implementation

    Implemented as a BPEL process using WebSphere Integration Developer. The execution environment is the WebSphere Process Server.

  - Reference

    12.3, “BPEL development” on page 361

- Group list Service component
  - Responsibilities

    Manage groups and members of groups.

  - Collaborations
    - Receives SIP messages from the Presence Service
    - Sends SIP messages to the Presence Service
    - Sends XCAP over Http messages to the Presence Service
  - Implementation

    IBM WebSphere Group List Server Component.

  - Reference

    8.5.1, “The role of Group List Management” on page 187

- Presence Service component
  - Responsibilities

    Manage the presence status of subscribers.

  - Collaborations
    - Receives SIP messages from the FindHelpSipServlet to get all active members for a group
    - Sends SIP messages to the FindHelpSipServlet
    - Sends SIP messages to the Group list Service to resolve the group to its members
    - Receives SIP messages from the Group list Service
    - Receives XCAP over Http messages from the Group list Service
    - Receives SIp messages from the PresenceClient about the presence status
  - Implementation

    IBM WebSphere Presence Server.

  - Reference

    8.4.2, “The Presence Management enabler” on page 184

- Location Service component
  - Responsibilities
    - Determine the current location of a subscriber in terms of geographical longitude and latitude)
    - Calculate the distance between the current location of subscriber and a given geographical location
  - Collaborations

    Provides a Web service, which is invoked by the FindHelpBpelProcess.
  - Implementation

    For our sample we have developed a location simulator. It is available in the additional material of this book. In a production environment this would be a commercially available Location Server.
  - Reference

    12.5, “The location simulator” on page 435
- DiameterRfService component
  - Responsibilities

    Diameter offline charging client.
  - Collaborations
    - Provide a Diameter Rf Web service interface, which is invoked by the FindHelpBpelProcess
    - Invoke the Diameter Rf interface of the ChargingCollection Function
  - Implementation

    IBM Connector Diameter component.
  - Reference

    8.3.2, “Diameter services” on page 178
- ChargingCollectionFunction component
  - Responsibilities
    - Collect charging data
    - Generate charging records
  - Collaborations

    Provides a Diameter Rf interface, that is called by the DiameterRfService.
  - Implementation

We have implemented a CCF simulator, which is available in the additional material of this book. In a production environment this component is implemented by a commercially available charging collection function.

– Reference

13.1, “Overview of the test environment” on page 440

ThirdParty Call Control component

– Responsibilities

- Establish a call between two subscribers
- Tear-down a call

– Collaborations

- Provides the Parlay X Third Party Call Control Web service, that is invoked by FindHelpBpelProcess
- Send SIP messages to the VoiceClient to establish a voice session
- Receive SIP messages from the VoiceClient

– Implementation

IBM WebSphere Telecom Web Services Server.

– Reference

8.6, “Telecom Web Services Server” on page 193

11.4.5 Component flow

The dynamic component interaction and message flows that make up the FindHelp application is presented in this section. Figure 11-9 on page 323 shows dynamic interactions between all the components in this scenario. Each interaction is labeled with a number ranging from 1 to 17. The numbers indicate different steps in the interaction. Figure 11-10 on page 326, Figure 11-11 on page 327 and Figure 11-12 on page 328 provide the component interaction diagrams for the scenario.

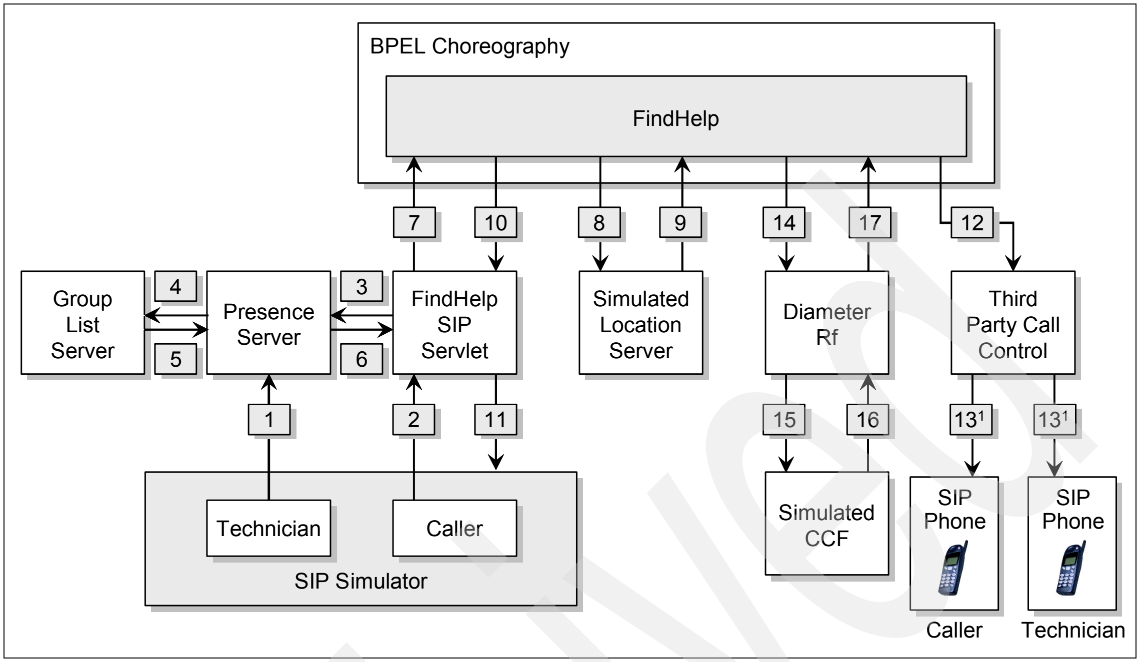


Figure 11-9 Component interaction for the FindHelp scenario.

> **Note:** It is assumed that the A-Party has already subscribed to the FindHelp service with the service provider, and therefore is allowed to access the service. It is also assumed that A-Party's and technician(s) devices are already powered-on and registered to the IMS Registration Subsystem (likely the CSCF). Finally only the successful use case is explained (at least one technician is available).

1. Technician(s) publishes his/her availability for the *LockOutService* by sending a SIP Publish message to the Presence Server. The message contains the state information about the technician.
2. A-Party invokes the *FindHelp* service on his device. He selects *LockOut Services* from the list of available topics proposed by the service provider, and clicks OK. The *FindHelp* Dialog application running in the A-Party's device initiates a SIP session with the *FindHelp* service by sending a SIP INVITE request message containing the topic selected by the A-Party, i.e. *LockOutServices*. The *FindHelp* service sends back a provisional SIP "100 Trying" response back to the device, followed by a SIP "200 OK" success response.

3. The FindHelp service subscribes to the presence status of the technicians in charge of the *LockOutServices* by sending a SIP SUBSCRIBE request to the Presence Server containing the corresponding group URI in the request in the “To” header of the SIP request. The Presence Server sends back a SIP “200 OK” success response.
4. The Presence server checks the REQUIRE header to see if the subscription may relate to a resource-list entity. If it is the case, then it checks if there are already previous subscription(s) for this group. If there are, then the current members information is used and a NOTIFY response is sent back to the device client with all the member presence information contained in a MIME Multipart message format. Otherwise, Presence Server sends a SIP SUBSCRIBE to the Group List Management Server since starting this point it wants to be notified about any change to the group content.
5. The Group List server sends back a SIP “200 OK” success response, then a SIP NOTIFY message containing the XCAP URL of the group. The Presence Server utilizes the XCAP protocol to retrieve the resource-list document containing the group members from the GLMS.
6. The Presence server composes a SIP NOTIFY requests containing current state of all group members, and sends the message to the *FindHelp* Service. Until subscription to the *FindHelp* service expires, the Presence Server will send a SIP NOTIFY request to the *FindHelp* service each time a change in either group content or group member state occurs.
7. The FindHelp service analysis the SIP NOTIFY request received from the Presence Server and computes a list of available technicians. It then invokes the FindHelp BPEL business process (implemented as a Web Service) and provides the identity of the A-Party and the list of available technicians as parameters to the request.
8. The FindHelp BPEL process starts a number of interactions with the Location Service:
   – A first interaction is started from the BPEL process to the Location Service to determine the current location of the A-Party. A-Party’s identity is provided as parameter to the request.
9. Depending to the number N of available technicians, (N-1) interactions is (are) started from the BPEL process to the Location Service to determine the distance between A-Party and the technician. Location Service sends back the distance information for each technician to the FindHelp BPEL process.
10. The FindHelp BPEL process computes the nearest technician, and sends back the technician identity to the FindHelp Service.
11. The FindHelp service computes a SIP INFO message containing the technician’s identity and sends the message to the A-Party’s device client. this information is displayed on A-Party’s device indicating that a SIP-Voice

call is being established with the technician. A-Party UA acknowledges the INFO message. A SIP BYE Message is sent by the *FindHelp* Service to close the SIP Session.

12. The FindHelp BPEL process invokes the ThirdParty Call Control component in order to establish a SIP voice call between A-Party and the technician. To do so it uses the Parlay X Web Service Interface.

13. The ThirdParty Call Control component, acting as a controller, establishes both legs of the voice call session by sending first SIP INVITE message to the the technician's User Agent (UA). Technician's phone rings, and technician answers. This results in a 200 RC OK that contains an offer, let's call it `offer1`. The controller needs to send an answer using a SIP ACK request, as mandated by RFC3261. To obtain the answer, it sends `offer1` to A-Party using a SIP INVITE message. A-Party's phone rings. When he answers, the 200 OK contains the answer to `offer1`, let's call it `answer1`, The controller sends an ACK to A-Party, and then passes `answer1` to the technician by sending an ACK message to the technician's UA. Because the offer was generated by the technician's UA, and the answer generated by A-Party's UA, the actual media session is between the technician and A-Party UAs. Refer to RFC3725 for more details and options for ThirdParty Call Control.

14. The FindHelp BPEL process invokes the DiameterRfService component using the Web Service Interface to charge A Party.

15. An Accounting Event is sent to the CCF simulator using Diameter protocol.

16. The CCF returns back an acknowledgment.

17. The DiameterRfService WS returns back a positive answer to the FindHelp BPEL process. The BPEL process is ended.

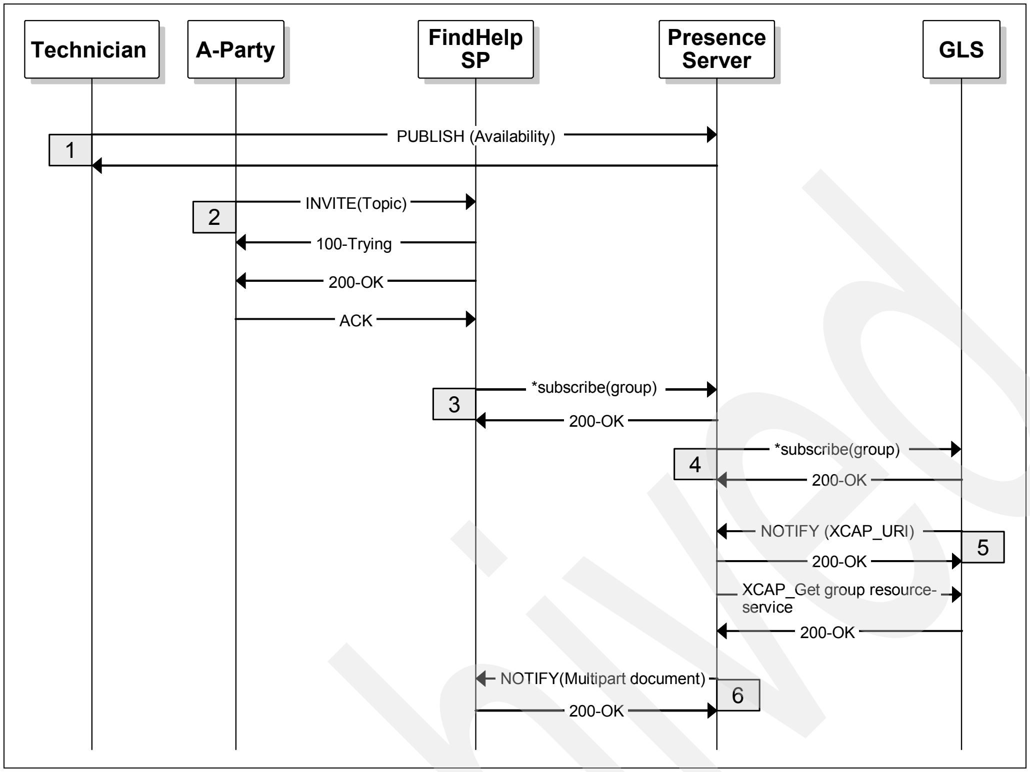


Figure 11-10 FindHelp Component Interaction Diagram (1/3)

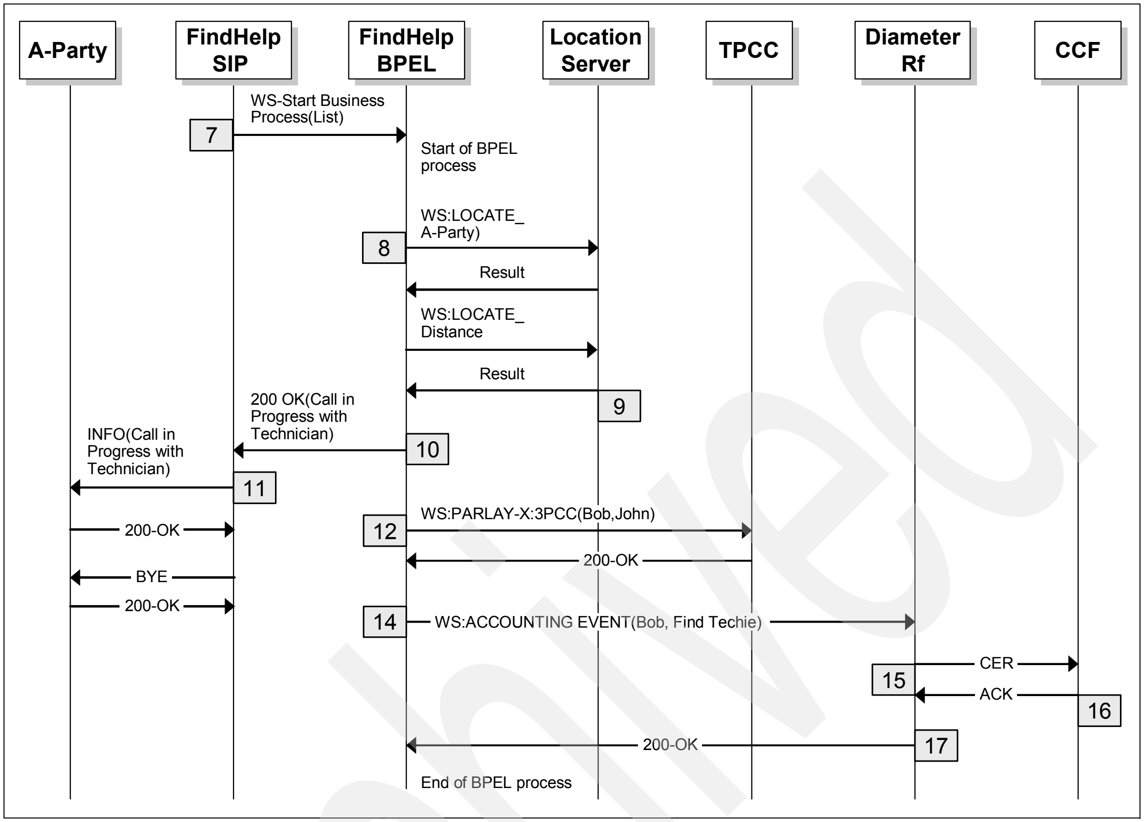


Figure 11-11 FindHelp Component Interaction Diagram (2/3)

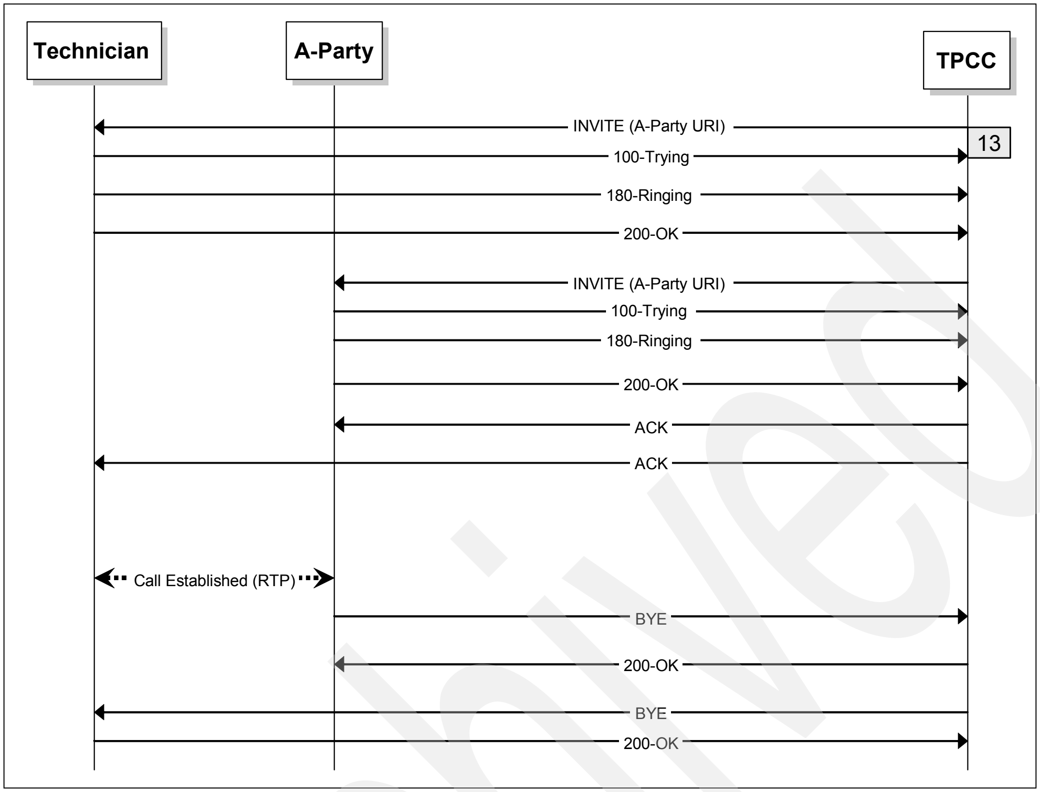


Figure 11-12 FindHelp Component Interaction Diagram (3/3)

11.5 SIP Servlet design

The SIP Servlet is the central control entity within the service. It acts as the control point for the device client, Presence server (including the Group List server) and the BPEL process running in Process server. The call flow is shown in Figure 11-13 on page 329.

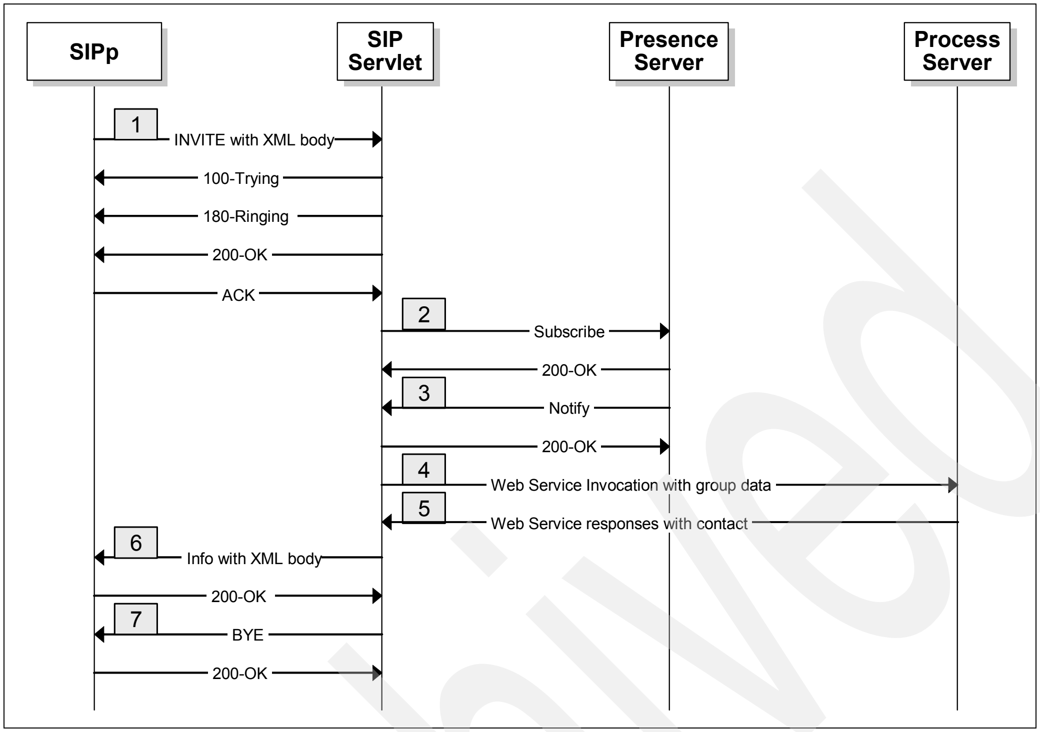


Figure 11-13 SIP and Web Service call flow for the SIP Servlet

These steps are a walkthrough of the call flow:

1. A SIP INVITE invokes the SIP Servlet. The SIP message includes an XML payload and details the group that the user requires assistance for.

> **Note:** In the sample scenario we implemented in this redbook, there is no integrated device client. Instead, it is simulated with a SIPp script and separate SIP phone. This means that the third party call established via the BPEL will need to have the SIP phone URI and not the SIPp URI. To allow this behavior the INVITE from address will correspond to the SIP phone, while the contact header will match the SIPp URI.

An example of the INVITE is presented in Example 11-1.

Example 11-1 Sample INVITE

```
INVITE sip:FindHelp@127.0.0.1:5065 SIP/2.0
Via: SIP/2.0/UDP 9.42.170.160:5068;branch=z9hG4bK4F58B3359B841
From: sipp <sip:sipphone@9.42.171.130>;tag=1
```

```
To: sut <sip:FindHelp@127.0.0.1:5065>
Call-ID: 1-6964@9.42.170.160
Cseq: 1 INVITE
Contact: sip:sipp@9.42.170.160:5068
Max-Forwards: 70
Subject: FindHelp
P-Charging-Vector:
icid-value=294_1124116770286@47.135.114.87;orig-ioi=scscf1@homedomain.c
om
Content-Type: text/xml
Content-Length:  112

<?xml version="1.0" encoding="UTF-8"?>
<FindHelp>
<group>sip:mytechies@itso.ral.ibm.com</group>
</FindHelp>
```

In the above example the SIP phone URI is sip:sipphone@9.42.171.130 and the SIPp is running at sip:sipp@9.42.170.160:5068. Once the message is received by the servlet the XML body of the INVITE is parsed to retrieve the SIP URI that corresponds to the group the end user requires assistance from for example sip:mytechnies@itso.ral.ibm.com.

2. To register a subscription to the Presence server for a certain URI the SIP To header of a subscribe message is used to identify the user/group the subscription relates to and the SIP URI for routing to the Presence server. If the subscription is regarding a group then the supported header must include `eventlist`. This informs the presence server that the request may be regarding a group and it should consult the group list server. The sample application only requires the current presence information as one notify and any future changes do not need to be forwarded. To accomplish this behavior the expires header is set to `0` notifying the Presence server that only a snapshot is required. An example of the SUBSCRIBE message is presented in Example 11-2.

Example 11-2 Sample SUBSCRIBE

```
SUBSCRIBE sip:service@localhost:5063 SIP/2.0
From: "sut"
<sip:FindHelp@127.0.0.1>;tag=0718355665374052_local.1150309179673_33_71
To: <sip:mytechies@itso.ral.ibm.com>
Call-ID: 2020986185856275@9.44.220.56
Max-Forwards: 70
CSeq: 2 SUBSCRIBE
Event: presence
Supported: eventlist
```

```
Accept: application/pidf+xml, application/rlmi+xml, multipart/related,
amultipart/signed, application/pkcs7-mime
Expires: 0
Via: SIP/2.0/UDP 9.44.220.56:5065;branch=z9hG4bK96810741518611
Contact: <sip:9.44.220.56:5065;transport=udp>
Content-Length: 0
```

3. The Presence Server will communicate with the Group List Server (GLS) to retrieve group information. On receipt of the group data from the GLS it queries the presence information for the members of the group and composes a single response. The response is a multipart mime, with the initial section detailing the group information and subsequent sections providing the presence information for each member of the group for whom valid data exists. An example of a NOTIFY is shown in Example 11-3.

Example 11-3 Sample NOTIFY

```
NOTIFY sip:9.44.220.56:5065;transport=udp SIP/2.0
From:
<sip:mytechies@itso.ral.ibm.com>;tag=47328183334644003_local.1150231806
857_194_189
To: "sut"
<sip:FindHelp@127.0.0.1>;tag=0718355665374052_local.1150309179673_33_71
Call-ID: 2020986185856275@9.44.220.56
Max-Forwards: 70
CSeq: 2 NOTIFY
Content-Type:
Multipart/Related;type="application/rlmi+xml";start="mytechies%40itso.r
al.ibm.com";boundary="boundary-112"
Content-Length: 1359
Event: presence
Subscription-State: terminated
Require: eventlist
Via: SIP/2.0/TCP 9.44.220.56:5063;branch=z9hG4bK991449639733644
Contact: <sip:9.44.220.56:5063;transport=udp>

--boundary-112
Content-ID: "mytechies@itso.ral.ibm.com"
Content-Type: application/rlmi+xml

<?xml version="1.0" encoding="UTF-8"?><list
xmlns="urn:ietf:params:xml:ns:rlmi"
cid="&quot;mytechies@itso.ral.ibm.com&quot;" fullstate="true"
uri="mytechies@itso.ral.ibm.com" version="0"><resource
uri="jochen@itso.ral.ibm.com"/><resource
```

```
uri="callum@itso.ral.ibm.com"><instance cid="callum@itso.ral.ibm.com"
id="1" state="active"/></resource><resource
uri="cameron@itso.ral.ibm.com"/><resource
uri="rebecca@itso.ral.ibm.com"><instance cid="rebecca@itso.ral.ibm.com"
id="1" state="active"/></resource><resource
uri="phil@itso.ral.ibm.com"/></list>
--boundary-112
Content-ID: callum@itso.ral.ibm.com
Content-Type: application/pidf+xml

<?xml version="1.0" encoding="UTF-8"?>
<presence entity="callum@itso.ral.ibm.com"
xmlns="urn:ietf:params:xml:ns:pidf">
<tuple id="1234560001">
<status>
<basic>open</basic>
</status>
<contact>sip:callum@9.42.171.135</contact>
</tuple>
</presence>

--boundary-112
Content-ID: rebecca@itso.ral.ibm.com
Content-Type: application/pidf+xml

<?xml version="1.0" encoding="UTF-8"?>
<presence entity="rebecca@itso.ral.ibm.com"
xmlns="urn:ietf:params:xml:ns:pidf">
<tuple id="1234560002">
<status>
<basic>open</basic>
</status>
<contact>sip:rebecca@9.42.171.135</contact>
</tuple>
</presence>

--boundary-112
```

4. Parsing of the NOTIFY body is completed by the SIP Servlet to determine what members of the group are currently available. This information is then submitted to Process server with the `To` address of the INVITE (Step 1). As the Third Party Call Control logic does not communicate with a Proxy for name resolution, it is not possible to pass `xxx@itso.ral.ibm.com` to Process server and successfully resolve to a valid endpoint. Therefore the content of the contact xml tag is used as the valid SIP endpoint for establishing the third

party call control. It is critical that a SIP phone is available at this SIP endpoint to allow the third party call control logic to establish a call. An example Web service request to the Process Server is shown in Example 11-4.

Example 11-4 Web Service Request Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope xmlns:q0="http://FindHelp/FindHelpInterface"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
      <q0:invokeCallBack>
         <addresses>
            <address>sip:rebecca@9.42.171.135</address>
            <address>sip:callum@9.42.171.135</address>
         </addresses>
         <originator>sip:sipphone@9.42.171.130</originator>
      </q0:invokeCallBack>
   </soapenv:Body>
</soapenv:Envelope>
```

5. The Process Server will query the location server to determine the closest person from the available resources. It then returns the SIP endpoint to the calling SIP Servlet. Further details regarding the behavior within the Process server can be found in 12.3, “BPEL development” on page 361. A Web service response from the Process server is shown in Example 11-5.

Example 11-5 Web Service Response Example

```
<soapenv:Envelope
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header />
   <soapenv:Body>
      <interface:invokeCallBackResponse
xmlns:interface="http://FindHelp/FindHelpInterface">
         <status>OK</status>
         <callee>sip:callum@9.42.171.135</callee>
      </interface:invokeCallBackResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

6. The response from the Process Server invocation is packaged into a SIP INFO message and sent to the originating SIPp client to be displayed to the end user. An example of the INFO can be seen in Example 11-6.

Example 11-6 INFO Example

```
INFO sip:sipp@9.44.220.56:5068 SIP/2.0
From: "sut"
<sip:FindHelp@127.0.0.1:5065>;tag=5359090010121218_local.1150309179673_
28_60
To: "sipp" <sip:sipp@9.42.171.130>;tag=1
Call-ID: 1-4476@9.44.220.56
Max-Forwards: 70
CSeq: 2 INFO
Content-Type: text/xml
Content-Length: 82
Via: SIP/2.0/UDP 9.44.220.56:5065;branch=z9hG4bK577258802603871

<?xml version="1.0"
encoding="UTF-8"?><FindHelp><ringing>sip:callum@9.42.171.135</ringing><
/FindHelp>
```

7. The SIPp client will display the SIP address to the caller and respond with a 200 OK. This notifies the SIP Servlet that the dialog can be shutdown. A SIP BYE message is generated by the server and sent to the client. An example of the SIP message can be seen in Example 11-7.

Example 11-7 Example BYE message

```
BYE sip:sipp@9.44.220.56:5068 SIP/2.0
From: "sut"
<sip:FindHelp@127.0.0.1:5065>;tag=5359090010121218_local.1150309179673_
28_60
To: "sipp" <sip:sipp@9.42.171.130>;tag=1
Call-ID: 1-4476@9.44.220.56
Max-Forwards: 70
CSeq: 3 BYE
Via: SIP/2.0/UDP 9.44.220.56:5065;branch=z9hG4bK159275582063324
Content-Length: 0
```

Note: The discussion in this section focused on the SIP Servlet implementation. It does not address the process for creating XML parsers and Web Service Java clients. Several publications have already covered these in detail. The following Web sites and URLs that provide relevant further information:

- All about JAXP, Part 1

  http://www-128.ibm.com/developerworks/xml/library/x-jaxp/
- All about JAXP, Part 2

  http://www-128.ibm.com/developerworks/xml/library/x-jaxp2/
- Invoking Web Services with Java clients

  http://www-128.ibm.com/developerworks/webservices/library/ws-javaclient/

All the source code for parsing and Web service code is included in the additional material in siputils.jar, and an illustration of the dependencies is shown in Figure 11-14 on page 335.

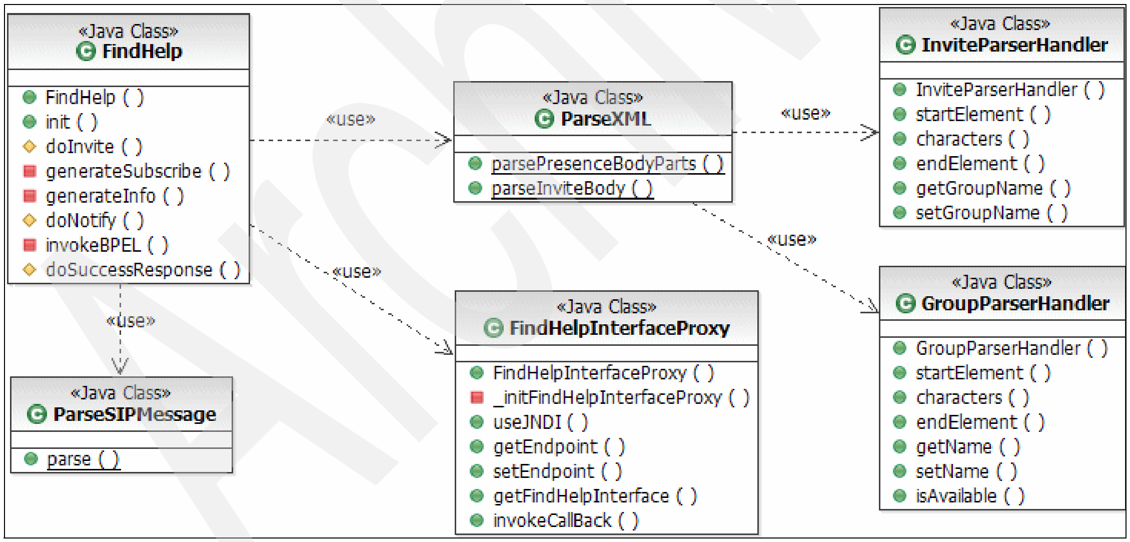


Figure 11-14 Class Diagram for the SIP Project

The FindHelp SIP Servlet uses three class within the utilize package:

- ParseSIPMessage

  When a multipart mime is received by the SIP Servlet a byte array is returned from the getContent method of the SipServletMessage class. The parsing of this method has to be handled manually by the application and therefore this class is used to separate the multipart mime into an array of XML strings for further processing.

- ParseXML

  ParseXML is used to parse the XML body of the INVITE to determine the SIP URI of the group to be contacted. This same class is utilized to parse the body FindHelpInterfaceProxy of the NOTIFY message from the presence server.

- FindHelpInterfaceProxy

  Provides a Java interface to invoke the BPEL. Assuming that the WSDL is not modified, this class should provided the functionality required including configuration of the service endpoint. However if the parameter names or methods of the WSDL representing the process server service is modified then a new Java client will be required.

11.5.1 BPEL design

The BPEL process exports a Web service that is invoked by the SIP Servlet. The invocation triggers the business process, which sequentially executes five process steps. Figure 11-15 on page 337 specifies the interactions between the BPEL process and the enabling services.

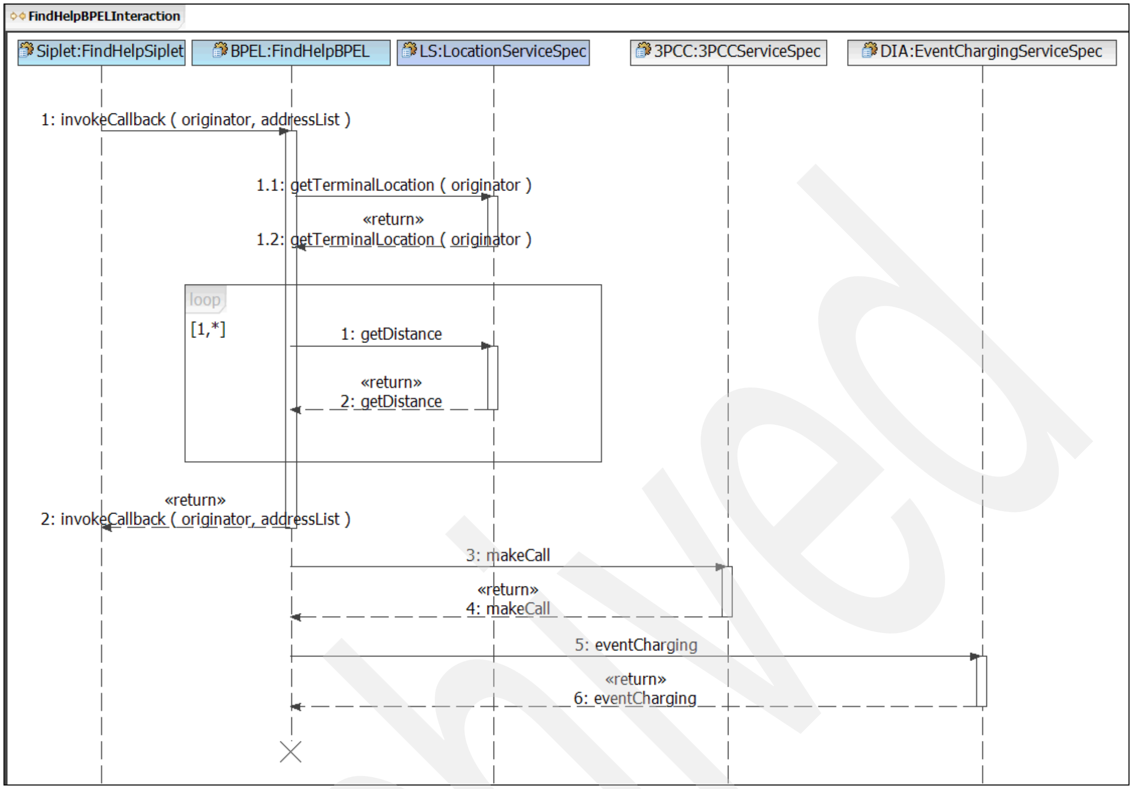


Figure 11-15 FindHelp BPEL Process Interaction Diagram

1. The process is started by the SIP servlet invoking the invokeCallBack operation. The Sip servlet passes the SIP address of the originator of the service and a list of SIP addresses of available technicians to the process.
2. The process invokes the getTerminalLocation operation to locate the originating (A-) party. The location service returns the geographical longitude and latitude of the A-party's current location.
3. Loop through the list of SIP addresses to determine for each address the distance between the addressee's location and the A-Party. Calculate the address, which is closest to the originator.
4. Return the closest address to the SIP servlet.
5. Invoke the makeCall operation of the ThirdPartyCallControl Web service to establish a call between the originator and the closest address.
6. Call the EventCharging Web service to charge for the use of the FindHelp service.

Applying the above to the BPEL principles described in 6.3, “Components” on page 123 we can now design a first assembly diagram (see Figure 11-16).

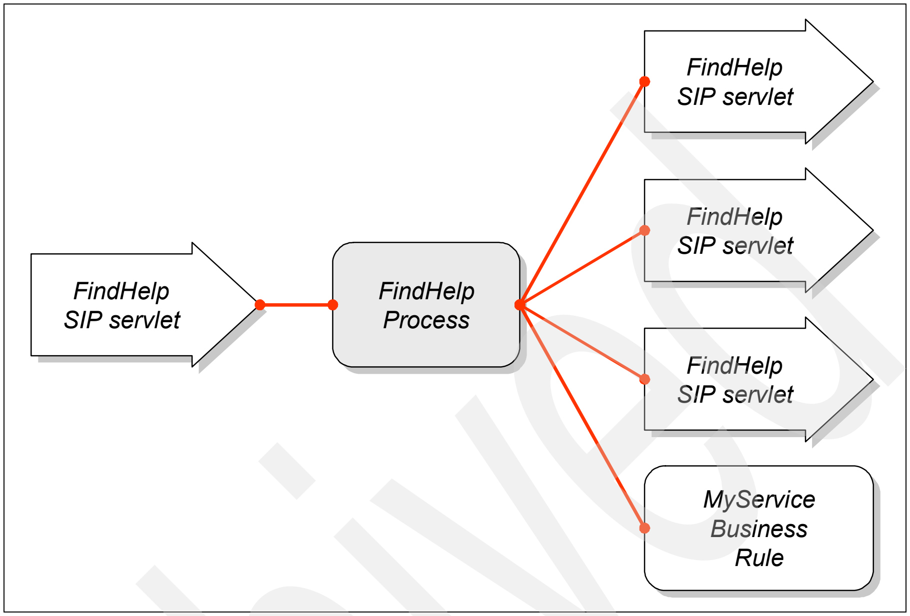


Figure 11-16 Module design

From the assembly diagram we can see, that we need to specify four interfaces:

- One Export Interface, to expose the service, which is provided to the SIP servlet.
- Three Import Interfaces, to describe the services required from the enablers.

Interfaces

The following interfaces are provided or required by the FindHelp module.

- FindHelpInterface

  This interface models the invocation interface for the FindHelp process.

  It provides a single operation invokeCallBack. Section 12.3.3, “Create the interface for the BPEL process” on page 368 shows how to create this interface.

Table 11-1 FindHelpInterface interface

Operation	Parameter type	Parameter	Data type
invokeCallBack	Input	addresses	AddressesBO

Operation	Parameter type	Parameter	Data type
	Input	originator	string
	Output	status	string
	Output	callee	string

- TerminalLocationImpl

  This interface is provided by Location service. It provides three operations, from which the FindHelp process invokes getTerminalLocation and getLocation.

Table 11-2 TerminalLocationImpl interface

Operation	Parameter type	Parameter	Data type
getLocation	Input	uri	anyURI
	Input	requestedAccuracy	int
	Input	acceptableAccuracy	int
	Output	getLocationReturn	LocationInfo
getTerminalDistance	Input	uri	anyURI
	Input	latitude	float
	Input	longitude	float
	Output	getTerminalDistanceReturn	int

- ThirdPartyCall

  This interface is provided by the Third Party Call Control Parlay X Web service, which is exposed by the WebSphere Telecom Web Services Server. Out of the four operations provided by this interface we only use the makeCall operation.

Table 11-3 ThirdPartyCall interface

Operation	Parameter type	Parameter	Data type
makeCall	Input	parameters	makeCall
	Output	result	makeCallResponse

- DiameterRfService_SEI

  The IBM WebSphere Diameter Enabler Component exposes an interface to generate charging events. Out of the five operations that are provided by this interface we only use the eventOfflineAcounting.

Table 11-4 DiameterRfService_SEI interface

Operation	Parameter type	Parameter	Data type
eventOfflineAccounting	Input	sessionId	string
	Input	recordNumber	int
	Input	userName	string
	Input	acctInterimInterval	int
	Input	destinationRealm	string
	Input	eventTimestamp	float
	Input	originStateID	int
	Input	act	Accounting
	Output	eventOfflineAccountingReturn	ACAResults

12

Implementing the IMS sample service

This chapter provides detailed descriptions for the development of *FindHelp*, the sample IMS service implemented in this redbook. It includes the development of the SIP Servlet for the FindHelp Service, the creation of the BPEL process using WebSphere Integration Developer and a location simulator which acts as a Location server.

This chapter contains the following:

- Implementation overview
- SIP Servlet development
- BPEL development
- The location simulator

12.1 Implementation overview

In this section, we create the SIP Servlets for the Registrar and LocalProxy. We also create the business logic for storing mappings between Addresses of Record provided by a User Agent and the User Agent's current contact information

The following steps detail the process for creating the new SIP Project required for the FindHelp service.

This section describes the implementation completed for the sample service. It is seperated into three sections:

- SIP Servlet development

  Describes the process for creating the SIP Servlet for the FindHelp Service. It details the steps for creating the sample code using the WebSphere Application Server Toolkit.

- BPEL Development

  Describes the BPEL process using the sample applications and provides information for creating the process flow within WebSphere Integration Developer.

- Location Simulator

  Describes the implementation of the location simulator at a high level to assist with debugging.

12.2 SIP Servlet development

The SIP Servlet is the central control entity within the FindHelp service. It acts as the control point for the device client, The development of the SIP Servlet is separated into five sections:

- Create a new SIP Project
- Create a new SIP Servlet
- Importing the associated Utility JAR file
- Complete the coding of the SIP Servlet
- Exporting the application for deployment

12.2.1 Create a new SIP project

You can create two different types of projects, a SIP or a Converged Project. You create a SIP Project when only SIP Servlets will be contained within the project.

And create a Converged Project when both SIP and HTTP servlets are required in the project. For the FindHelp service, only SIP Servlets are required. The following steps are for the creation of a SIP Project.

1. If the Application Server Toolkit V6.1 is not already started, select **Start** → **All Programs** → **IBM WebSphere** → **Application Server Toolkit V6.1** → **Application Server Toolkit**.
2. Select **File** → **New** → **Project**.

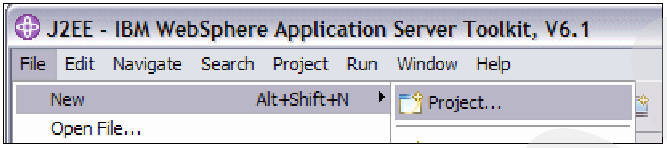


Figure 12-1 Defining a new SIP project within AST

3. Select **SIP** → **SIP Project**.

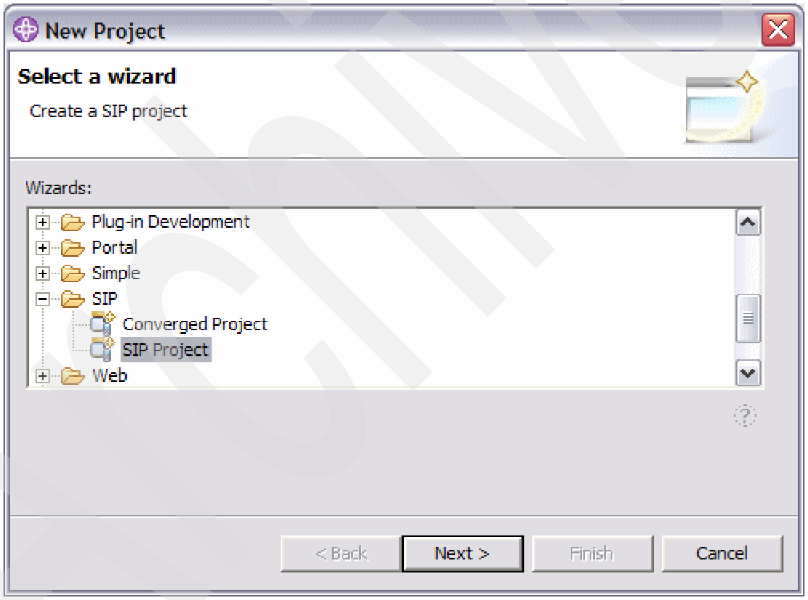


Figure 12-2 Select SIP Project from the New Project wizard

4. Click **Next**.
5. Define the properties of the new SIP Project.
   a. Enter `FindHelp` as the Project Name.
   b. Select `WebSphere Application Server v6.1` as the Target runtime.

6. Click **Finish**.

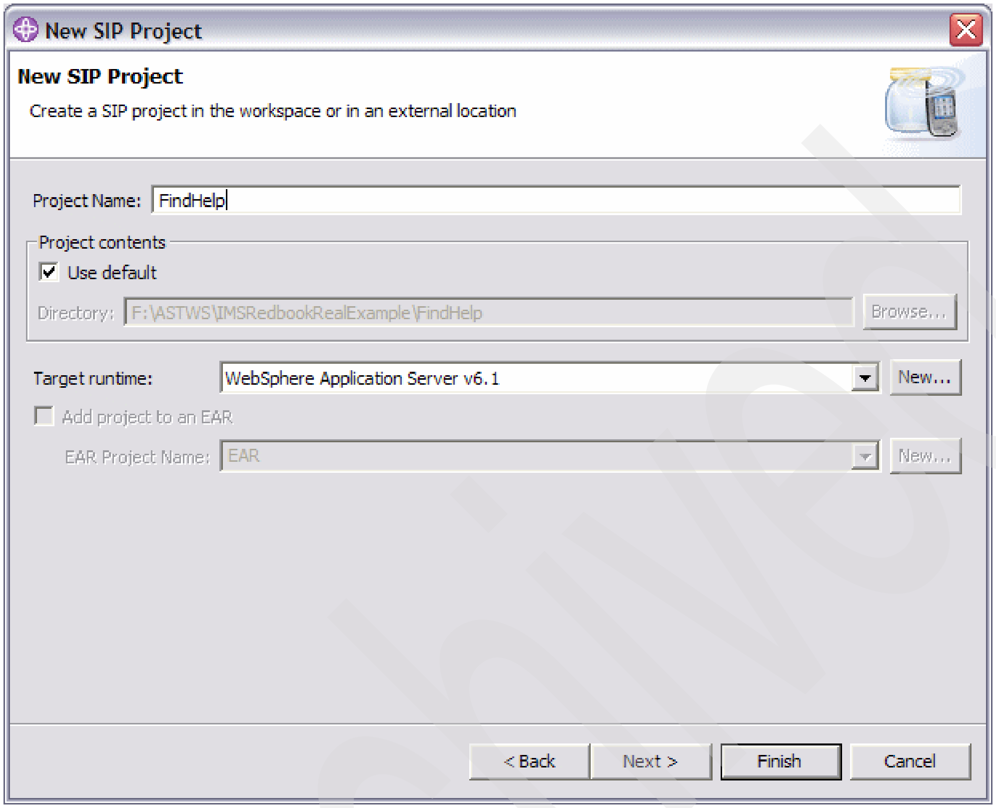


Figure 12-3 Defining the properties of the new SIP Project

A new SIP project will be created and you can find the newly created project under **Other Projects** → **FindHelp**.

12.2.2 Create a new SIP Servlet

Having created the SIP Project, you are now ready to create the SIP Servlet for the FindHelp service. The steps for creating the SIP Servlet are as follows:

1. Right-click the FindHelp application under **Other Projects** → **FindHelp**.
2. Select **New** → **Other**.
3. Select **SIP** → **SIP Servlet**.
4. Click **Next**.

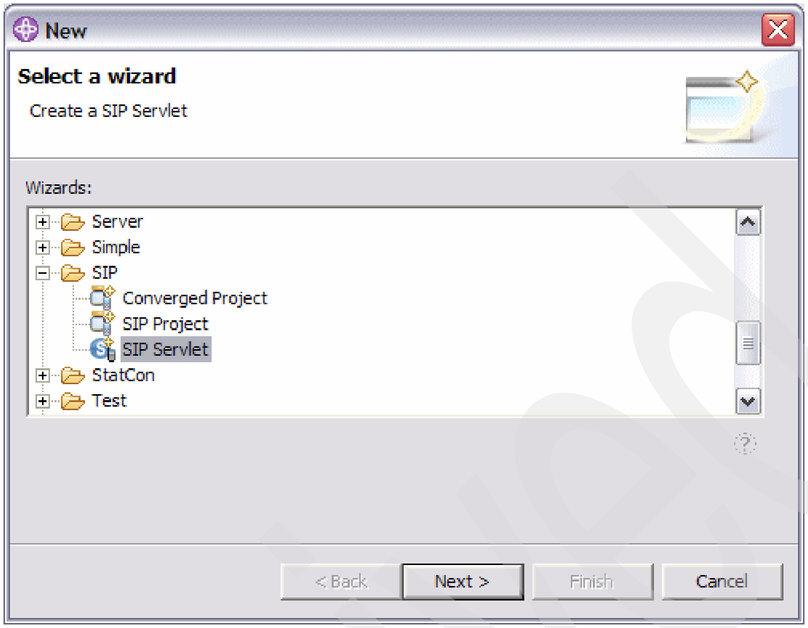


Figure 12-4 Select SIP Servlet from the creation wizard

5. In the Specify class file destination window fill in the following:
   a. Java Package field: `com.ibm.itso`
   b. Class name: `FindHelp`
6. Click **Next**.

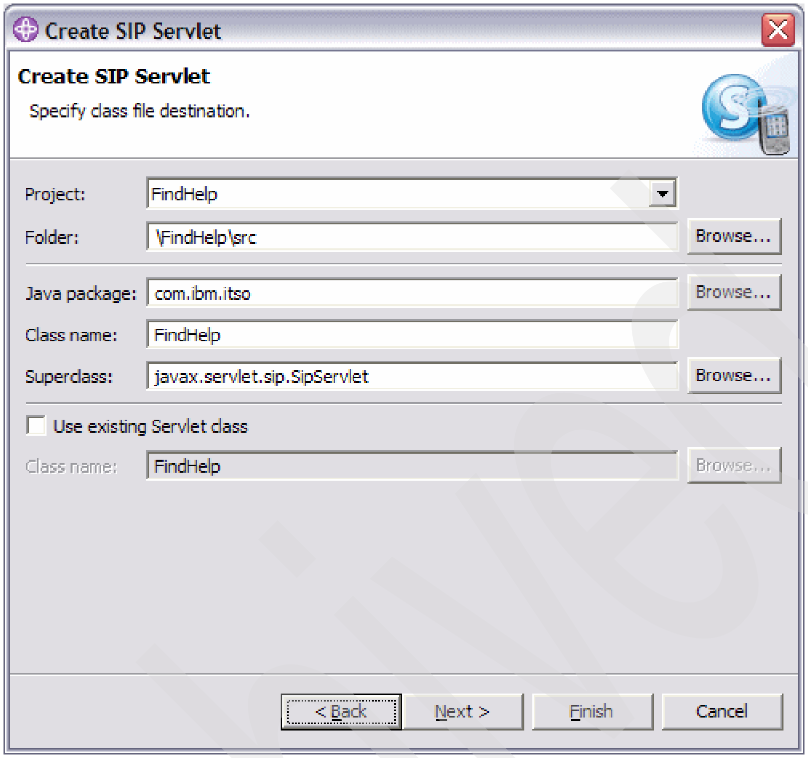


Figure 12-5 Define the SIP Servlet name and package

7. The servlet deployment descriptor specific information dialog will appear similar to Figure 12-6 on page 347.

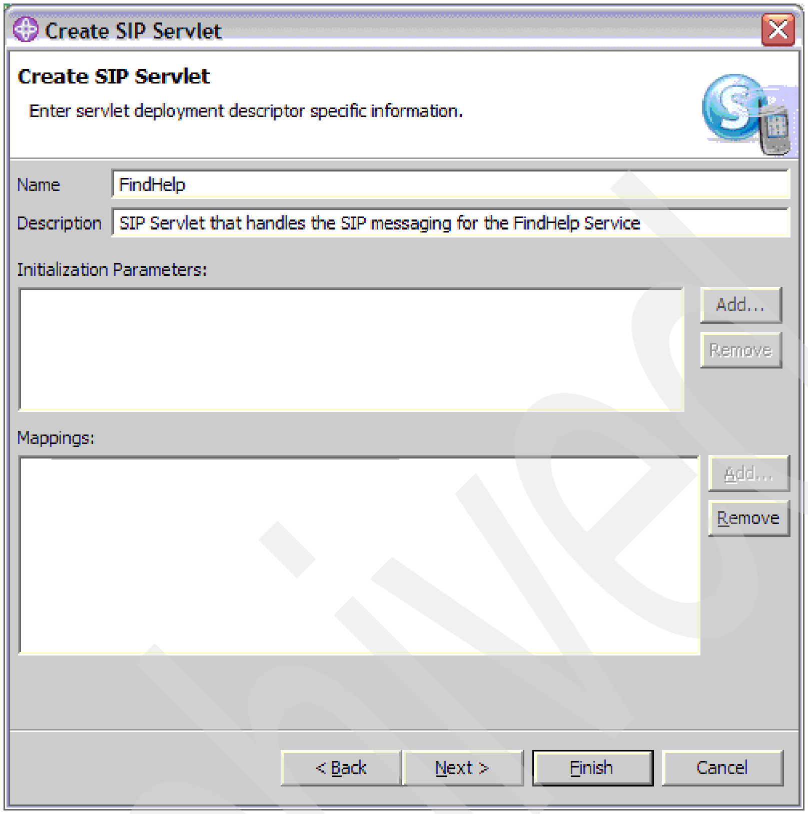


Figure 12-6 Servlet deployment descriptor specific information - add initialization parameters

8. Click **Add** beside the initialization Parameters box.
9. In the Initialization Parameters window similar to Figure 12-7 on page 348, fill in the following values:
   a. Name: `PresenceServerHostname`
   b. Value: `localhost` (or you may the hostname of the presence server)
   c. Description: `The hostname of the Presence Server`
10. Click **OK**.

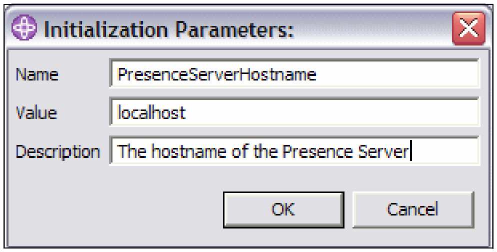


Figure 12-7 Initialization Parameters

11.Repeat Steps 8, 9 and 10 for the following parameters:

Table 12-1 Additional initialization parameters to be created

Name	Value	Description
PresenceServerPortNumber	5063	Port number for the presence server
ProcessServerURI	http://localhost:9080/FindHelpWeb/sca/setupcall	The URI of the BPEL process to be invoked by the SIP Servlet

> **Note:** You may have to use different values for your environment.

12.Click **Add** beside the Mappings box to enter the mapping information.

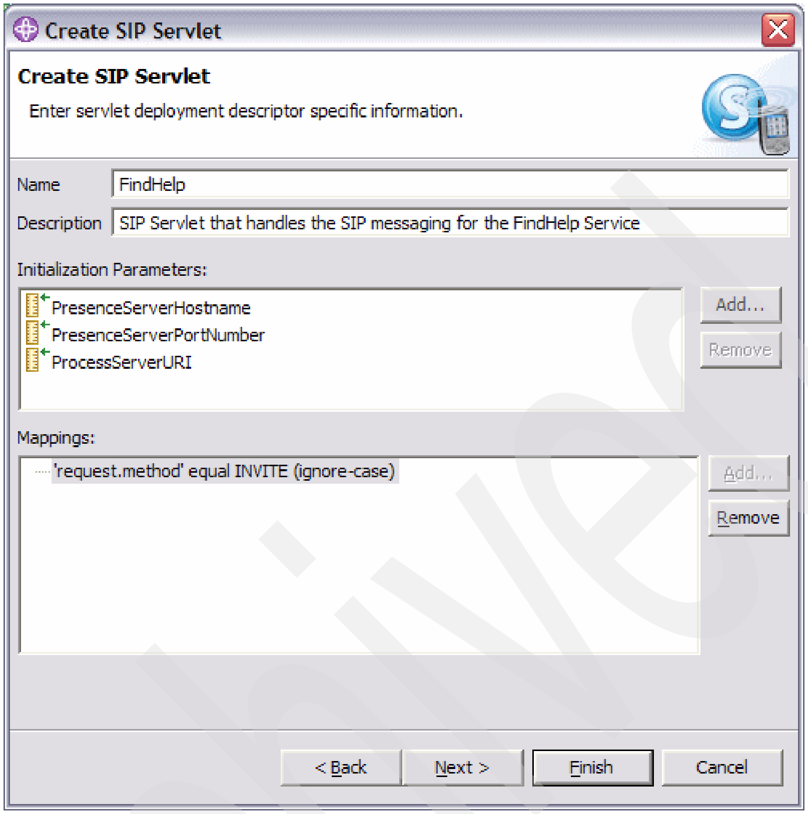


Figure 12-8   Servlet deployment descriptor specific information

13. In the pop-up menu, select **Condition**.
14. Fill the Add Mapping Condition window with the following values:
    a. Condition: `Equal`
    b. Variable: `request.method`
    c. Value: `INVITE`
15. Click **OK**.

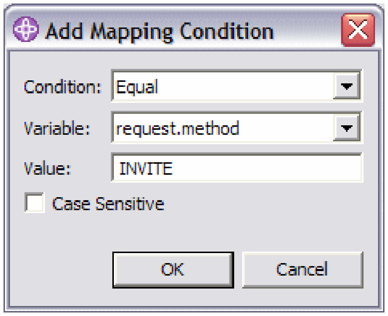


Figure 12-9 Add Mapping Condition for the SIP Servlet

The completed Servlet deployment descriptor specific information window will appear as in Figure 12-8 on page 349.

16. Click **Next**.
17. Specify the method stubs to generate by clicking the following method check boxes:
    a. doInvite
    b. doNotify
    c. doSuccessResponse
18. Click **Finish**.

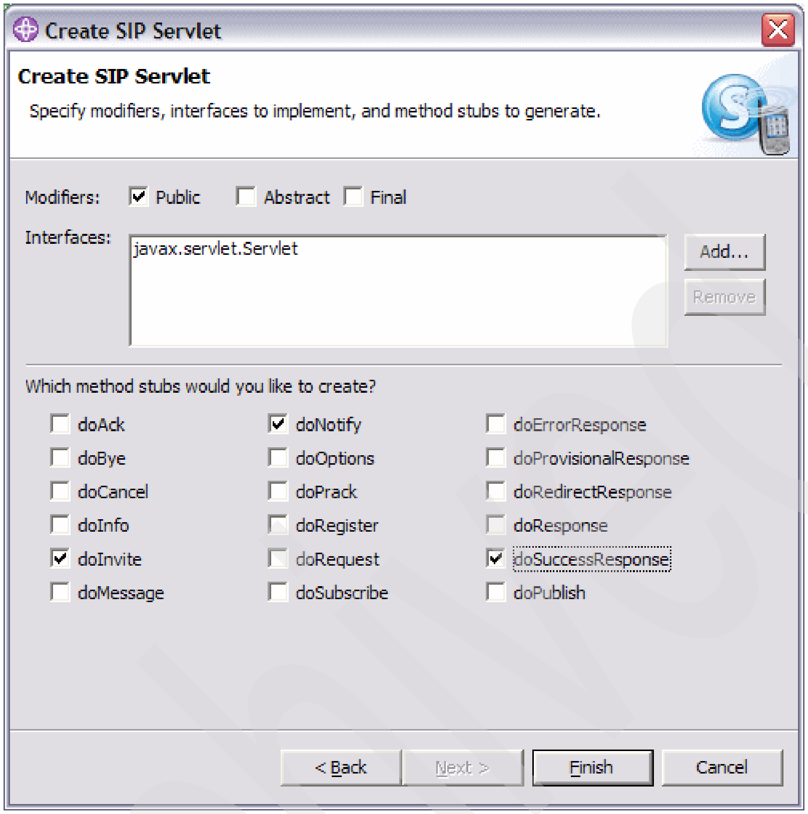


Figure 12-10 Method selection for the SIP Servlet

Import the associated Utility JAR file

The following details the process for importing the utilizes jar file into the new project, which will enable the SIP Servlet to access the required classes:

1. Expand **Other Project** from the Project Explorer.
2. Expand the project **FindHelp**.
3. Right-click **src**.
4. Select **Import** from the pop-up menu.
5. Select **Archive file**.
6. Click **Next**.
7. Click **Browse** and navigate to `siputils.jar` file.
8. Click **Open**.
9. Click **Finish** to complete the import.

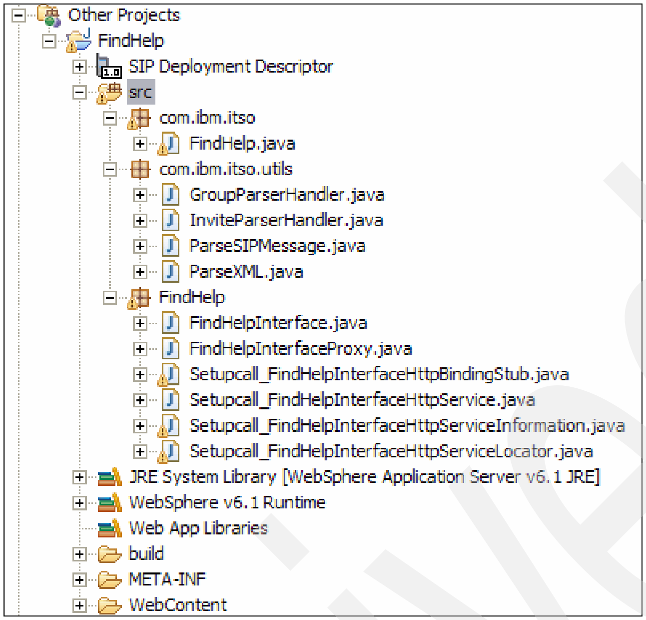


Figure 12-11 Importing siputils.jar

12.2.3 Complete the SIP Servlet code

In order to create a fully functional FindHelp SIP Servlet, complete the code generated by the AST. Here are the steps for completing the FindHelp SIP Servlet:

1. Open the new FindHelp SIP Servlet:
   a. Expand **Other Projects** → **src** → **com.ibm.itso**.
   b. Double-click **FindHelp.java**.
2. There are a number of external classes that are utilized in the SIP servlet and these need to be added as import statements. Append the list in Example 12-1 to the current import statements.

Example 12-1 Import statements required

```
import java.io.ByteArrayInputStream;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.logging.Logger;
import javax.activation.DataSource;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
```

```
import javax.servlet.sip.Address;
import javax.servlet.sip.ServletParseException;
import javax.servlet.sip.SipFactory;
import javax.servlet.sip.SipSession;
import javax.servlet.sip.SipURI;
import javax.servlet.sip.URI;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.rpc.holders.StringHolder;
import org.xml.sax.SAXException;
import FindHelp.FindHelpInterfaceProxy;
import com.ibm.itso.utils.ParseSIPMessage;
import com.ibm.itso.utils.ParseXML;
```

3. Copy and customize the class wide variables shown in Example 12-2.

Example 12-2 Class variables for the SIP Servlet

```
public class FindHelp extends javax.servlet.sip.SipServlet implements
javax.servlet.Servlet {
private static final long serialVersionUID = 1L;
private static final int portNumberDefault = 5060;
private static String presenceServerPortNumber = (new
Integer(portNumberDefault)).toString();
private static String presenceServerHostname = "localhost";
private static String processServerURI =
"http://localhost:9080/FindHelpWeb/sca/setupcall";
private static String originalSIPSession =
"FindHelp.originalSIPSession";
private static String to = "FindHelp.to";
private Logger log = null;
private ServletContext servletContext = null;
```

4. A number of the variables defined in the previous step need to be initialized. Similar to HTTP servlets, SIP Servlets have an init method that runs during the creation of the servlet. This method is used to read in values in the deployment descriptor which are then stored in the class variables. Examples of such variables are PresenceServerPortNumber, PresenceServerHostname and ProcessServerURI defined in 12.2.2, “Create a new SIP Servlet” on page 344. The logger is initialized and the servlet context is stored so additional SIP dialogs can be created as required. The example code is shown in Example 12-3 on page 354.

Example 12-3 init() example code

```
public void init(ServletConfig arg0) throws ServletException
{
   String className = this.getClass().getName();
   log = Logger.getLogger(className);
   String localPortNumber =
arg0.getInitParameter("PresenceServerPortNumber");
   String localHostname =
arg0.getInitParameter("PresenceServerHostname");
   String localProcessServer =
arg0.getInitParameter("ProcessServerURI");
   if(localPortNumber != null)
   {
      presenceServerPortNumber = localPortNumber;
   }
   if(localHostname != null)
   {
      presenceServerHostname = localHostname;
   }
   if(localProcessServer != null)
   {
      processServerURI = localProcessServer;
   }
   servletContext = arg0.getServletContext();
   log.info("init()
presenceServerPortnumber="+presenceServerPortNumber+"
presenceServerHostname="+presenceServerHostname);
   super.init(arg0);
}
```

5. The service will start with an INVITE message from the SIPp client. This message needs to be processed by the SIP Servlet within the `doInvite` method. The example code for the `doInvite` implementation can be seen in Example 12-4. The code retrieves the content type of the body, verifies that it is as expected text/xml and retrieves the body as a string. The body is then parsed by the utility classes to retrieve the group name. A SIP dialog is established with a `180 ringing` followed by `200 OK`. The originating SIP URI is stored within the application session for later retrieval (so the outbound INFO can be sent). Finally an outbound subscribe is generated in a new dialog to the presence server with the responding group's SIP URI. The logic for the subscribe message is completed in the generateSubscribe method.

Example 12-4 doInvite() example code

```
protected void doInvite(SipServletRequest arg0) throws
ServletException, IOException
{
   log.info("doInvite ENTRY()");
   String contentType = arg0.getContentType();
   String contentBody = null;
   String groupName = null;
   if(contentType!= null && contentType.equalsIgnoreCase("text/xml"))
   {
      contentBody = (String)arg0.getContent();
      log.info("contentBody='"+contentBody+"'");
      groupName = ParseXML.parseInviteBody(contentBody);
      log.info("groupName="+groupName);
   }
   log.info("contentType='"+contentType+"'");
   SipServletResponse ringing = arg0.createResponse(180);
   ringing.send();
   log.info("completed 180 send");
   SipServletResponse complete = arg0.createResponse(200);
   complete.send();
   log.info("completed 200 send");
   arg0.getApplicationSession().setAttribute(to, arg0.getFrom());
   boolean rc = generateSubscribe(arg0, groupName);
   log.info("generateSubscribe rc="+rc);
}
```

6. The class will report error for the missing method `generateSubscribe`. This code can be seen in Example 12-5 on page 356. The servlet needs to establish a new dialog with the Presence server, therefore the `servletContext` is used to retrieve the SipFactory. It is used first to create a new SIP address representing the group and secondly to create the SUBSCRIBE request. The Presence server requires several headers to be defined in the subscribe for the correct behavior to be observed:
   - **Event**: This is set for all Presence server requests to `presence`
   - **Supported**: Is used by the Presence server to determine if the URI being subscribed to is a group URI or individual. If this setting is not included the presence server will not communicate with group list server. Therefore this is included in the sample code.
   - **Accept**: Details the format of the body that the servlet will accept in response to the subscribe (for example in subsequent notify messages).
   - **Expires**: Is set to zero, and notifies the Presence server that the servlet requires a snapshot of the current status (for example, a single response)

and does not want to subscribe to presence information for an extended period of time.

The SipFactory is used to create a new SipURI corresponding to the Presence server endpoint and the `setRequestURI` method called to define this for the subscribe request. The request is then sent to the Presence server. Later in the service an INFO message will be sent to the originator of the service to notify them of who will contact them, thereby storing the original SIP session in the new SIP session for later retrieval.

Example 12-5 generateSubscribe() example code

```
private boolean generateSubscribe(SipServletRequest arg0, String
groupSIPAddress) throws IOException, ServletParseException
{
   log.info("generateSubscribe ENTRY()");
   SipFactory sipFactory = (SipFactory)
servletContext.getAttribute("javax.servlet.sip.SipFactory");
   Address sipGroup = sipFactory.createAddress(groupSIPAddress);
   SipServletRequest subscribeRequest =
sipFactory.createRequest(arg0.getApplicationSession(), "SUBSCRIBE",
arg0.getTo(), sipGroup);
   subscribeRequest.setHeader("Event", "presence");
   subscribeRequest.setHeader("Supported", "eventlist");
   subscribeRequest.setHeader("Accept", "application/pidf+xml,
application/rlmi+xml, multipart/related, amultipart/signed,
application/pkcs7-mime");
   subscribeRequest.setExpires(0);
   SipURI sipURI =
sipFactory.createSipURI("service",presenceServerHostname);
   int sipPort = portNumberDefault;
   try
   {
      sipPort = (new Integer(presenceServerPortNumber)).intValue();
   }
   catch (NumberFormatException e)
   {
      log.warning("The port number for the Group List Server is not
defined correctly, it is currently set to'"+presenceServerPortNumber+"'
will it should be an int");
      e.printStackTrace();
   }
   sipURI.setPort(sipPort);
   subscribeRequest.setRequestURI(sipURI);
   subscribeRequest.send();
```

```
   log.info("generateSubscribe EXIT() sent to
"+subscribeRequest.getTo().toString()+" from
"+subscribeRequest.getFrom());
   subscribeRequest.getSession().setAttribute(originalSIPSession,
arg0.getSession());
   return true;
}
```

7. Once the SUBSCRIBE has been processed by the Presence server, a notify will be sent back to the servlet with a multipart mime with the initial part detailing the members of the group and an additional part for each member of the group that the Presence server holds valid status.

   Immediately after the notify is received a 200 `OK` response is generated and sent to the Presence server to ensure that any delay in processing the XML body does not cause resends by the Presence server.

   The body and content type of the request is retrieved and assuming the body is a byte array (which it will be since a multipart mime is passed to the servlet as a byte array), it is converted into a string for easier manipulation.

   After retrieving the string representation of the body, the utility class `ParseSIPMessage` is used to retrieve an ArrayList that stores each part of the message as a separate string (one for the group information, and zero or more items corresponding to the XML presence information available for the members of the group).

   This ArrayList needs to be processed by an XML parser to determine which members presence information exist, and if they are currently available to receive a call. The `ParseXML` utility class is used to parse the XML contained in the ArrayList and return an ArrayList with the available members SIP URIs. The originators SIP URI is retrieved from the application session (this was stored during the processing of the INVITE), and passed with the list of available members to the BPEL for processing.

   BPEL will then determine the member of the list closest to the originator and passes this back as a string. This string will be encoded into an XML string and passed with the request object to the `generateInfo` method that is responsible for sending the SIP INFO to the originator.

   The sample code for the `doNotify` can be found in Example 12-6 on page 357.

Example 12-6 doNotify() sample code

```
protected void doNotify(SipServletRequest arg0) throws
ServletException, IOException
{
   log.info("doNotify() ENTRY");
```

```
   SipServletResponse resp = arg0.createResponse(200);
   resp.send();
   Object content = arg0.getContent();
   String contentType = arg0.getContentType();
   String body = null;
   if(content != null && contentType != null)
   {
      log.info("doNotify() content.getClass().getCanonicalName()="+
content.getClass().getCanonicalName());
      log.info("doNotify() conentType="+contentType);
      if (content instanceof byte[])
      {
         body = new String((byte[]) content);
      }
      log.info("body="+body);
      if(body!=null)
      {
         ArrayList xmlBodyParts = ParseSIPMessage.parse(body,
contentType);
         ArrayList availableMembers =
ParseXML.parsePresenceBodyParts(xmlBodyParts);
         Address toAddress =
(Address)arg0.getApplicationSession().getAttribute(to);
         String memberToBeContacted = invokeBPEL(availableMembers,
toAddress.getURI().toString());
         log.info("memberToBeContacted="+memberToBeContacted + " from
the list of availableMembers="+availableMembers.toString());
         String bodyInfo = "<?xml version=\"1.0\"
encoding=\"UTF-8\"?><FindHelp><ringing>"+memberToBeContacted+"</ringing
></FindHelp>";
         generateInfo(arg0, bodyInfo);
      }
   }
   super.doNotify(arg0);
}
```

8. Two errors will be reported regarding `invokeBPEL` and `generateInfo` being undefined methods. See Example 12-7 on page 359 for sample code for the `invokeBPEL` method. The code creates a `FindHelpInterfaceProxy` object that allows communication to the BPEL Web service. It takes two input parameters and returns two output parameters. The signature of the method is:

```
public void invokeCallBack(
   java.lang.String[] addresses,
```

```
java.lang.String originator,
javax.xml.rpc.holders.StringHolder status,
javax.xml.rpc.holders.StringHolder callee)
```

Where, `addresses` contains a list of the available members of the group, the `originator` holds a string representation of the caller's SIP URI and the two `stringHolders` store the output parameters `status` and `callee`.

Example 12-7 invokeBPEL() sample code

```
private String invokeBPEL(ArrayList availableGroupMembers, String
originator)
{
   log.info("invokeBPEL() ENTRY, originator="+originator+"
availableGroupMembers="+availableGroupMembers.toString());
   FindHelpInterfaceProxy bpel = new FindHelpInterfaceProxy();
   bpel.setEndpoint(processServerURI);
   log.info("invokeBPEL() set URI for BPEL to="+processServerURI);
   String[] availableMembers = new
String[availableGroupMembers.size()];
   availableGroupMembers.toArray(availableMembers);
   StringHolder status = new StringHolder();
   StringHolder callee = new StringHolder();
   try
   {
      bpel.invokeCallBack(availableMembers, originator, status,
callee);
      log.info("invokeBPEL() Callee returned='"+callee.value+"
status="+status.value);
      return callee.value;
   } catch (RemoteException e) {
      log.info("RemoteException occurred");
      e.printStackTrace();
   }
   return null;
}
```

9. Sample code for the `generateInfo` method can be seen in Example 12-8 on page 360. The code retrieves the original SIP session from the current session (this was stored in Example 12-5 on page 356), which it uses to generate a new outbound INFO with the body attached, it then sends the request.

Example 12-8 generateInfo() sample code

```
private boolean generateInfo(SipServletRequest arg0, String body)
throws IOException, ServletParseException
{
   log.info("generateInfo ENTRY()");
   SipSession originalSipSession =
(SipSession)arg0.getSession().getAttribute(originalSIPSession);
   SipServletRequest infoRequest =
originalSipSession.createRequest("INFO");
   infoRequest.setContent(body, "text/xml");
   infoRequest.send();
   log.info("generateInfo EXIT() sent");
   return true;
}
```

10. Once the client receives the INFO message it responds with a 200 OK, signaling to the servlet to send a SIP BYE to end the session. The sample code for the `doSuccessResponse` method is shown in Example 12-9. It receives the successful response (200 OK), verifies that it corresponds to an INFO request, it then creates and sends a new BYE request.

Example 12-9 doSuccessResponse() sample code

```
protected void doSuccessResponse(SipServletResponse arg0) throws
ServletException, IOException {
   log.info("doSuccessResponse() ENTRY METHOD="+arg0.getMethod());
   if(arg0.getMethod().equals("INFO"))
   {
      SipServletRequest bye = arg0.getSession().createRequest("BYE");
      bye.send();
   }
   super.doSuccessResponse(arg0);
}
```

12.2.4 Export the application for deployment

The completed application has to be exported as a SAR file for it to be deployed in the WebSphere Application Server 6.1 runtime.

1. Right-click **FindHelp** application and select **Export**.
2. Select **SAR File**.
3. Click **Next**.

4. Ensure that **FindHelp** is selected for the SIP project, and that a valid destination is entered (for example, c:\temp\FindHelp.sar).
5. Click **Finish**.

The SIP Servlet development is now complete and ready for deployment onto a WebSphere Application Server 6.1 runtime environment.

12.3 BPEL development

The BPEL process choreographs enablers (including the Location and Charging services) and implements the core business logic of the FindHelp sample application. In this section, we provide a step-by-step description of the development of the integration module.

> **Note:** The focus of this section is on the development of BPEL processes. A brief introduction to WebSphere Integration Developer is presented in Chapter 6, "IBM WebSphere Integration Developer" on page 119.
>
> For more information about the WebSphere Integration Developer and the WebSphere Process Server, refer to the redbook *Getting started with WebSphere Integration Developer and WebSphere Process Server*, SG24-7130.

12.3.1 Create a new business integration module

To begin building the FindHelp application, you need to create a new module called FindHelp as follows:

1. Select **File** → **New** → **Other**.
2. Select **Module** as shown in Figure 12-12 on page 362.
3. Click **Next**.

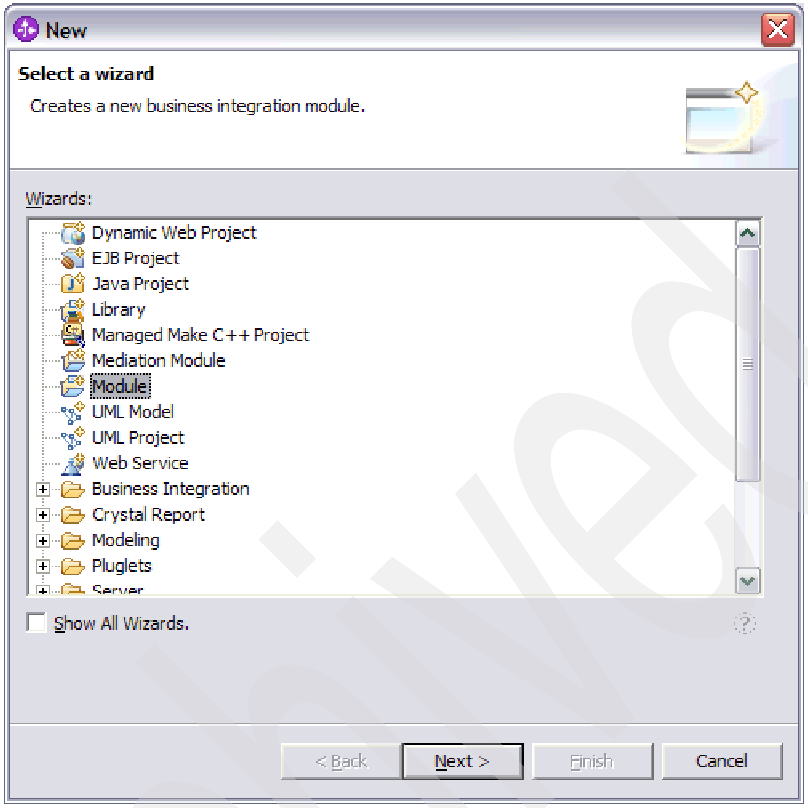


Figure 12-12 Select a wizard

4. Fill in the New Module window with values as in Figure 12-13 on page 363:
   a. Module Name: `FindHelp`
   b. Module location: Click the check box for Use default.

> **Note:** We used the default location, you can a different location for your solution.

5. Click **Finish**.

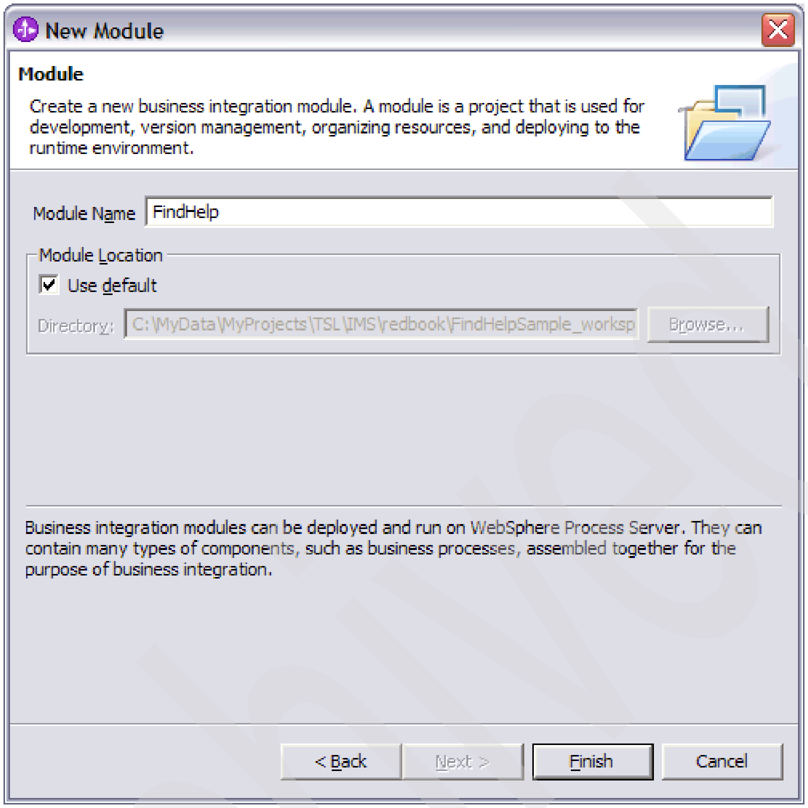


Figure 12-13 The FindHelp module

> **Note:** When you click **`Finish`** and the correct perspective is not already set, then you are prompted with a dialog box asking if to change the perspective. Click **`Yes`** to switch to the Business Integration Perspective.

6. Expand the FindHelp module.

   The structure and artifacts will be displayed as shown in Figure 12-14 on page 364. The organization of the structure maps to the building blocks of WebSphere Process Server. For an introductary discussion of these building blocks see 6.3.1, “Business Integration perspective and views” on page 126.

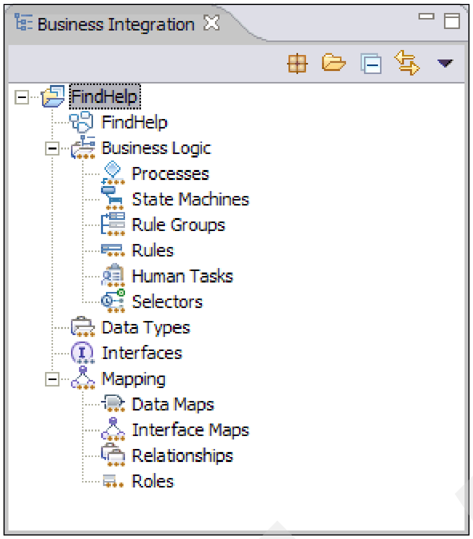


Figure 12-14 FindHelp module

12.3.2 Create the business object

We need to define the AddressesBO business object which is required for the address parameter in the FindHelp interface. The AddressesBO contains an array of strings.

> **Note:** All other business objects are created when the respective WSDL files are imported.

To create a new data type execute the following steps:

1. In the Business Integration perspective.
2. Right-click the folder **Data Types**.
3. Select **New** → **Business Object**.

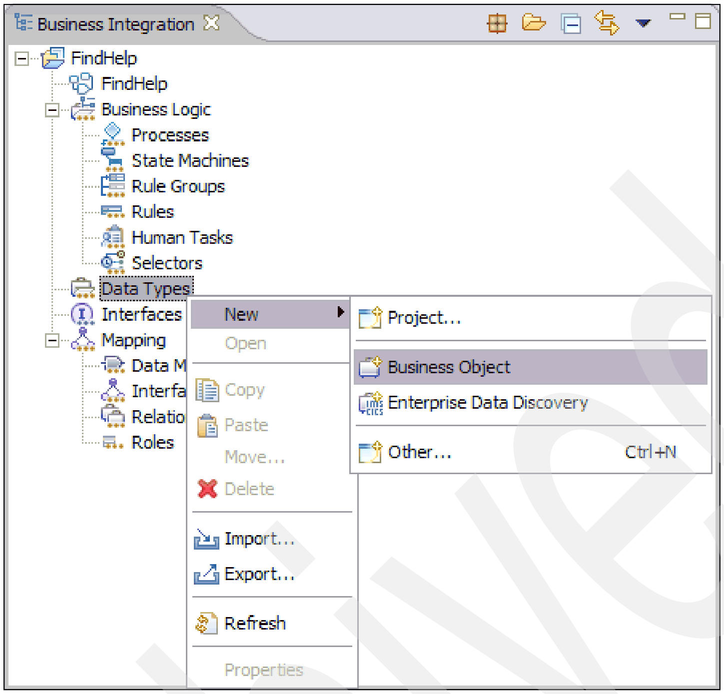


Figure 12-15 New business object

4. This starts the New Data Object Wizard.
5. Enter AddressesBO for the name for the business object as shown in Figure 12-16 on page 366.
6. Click **Finish**.

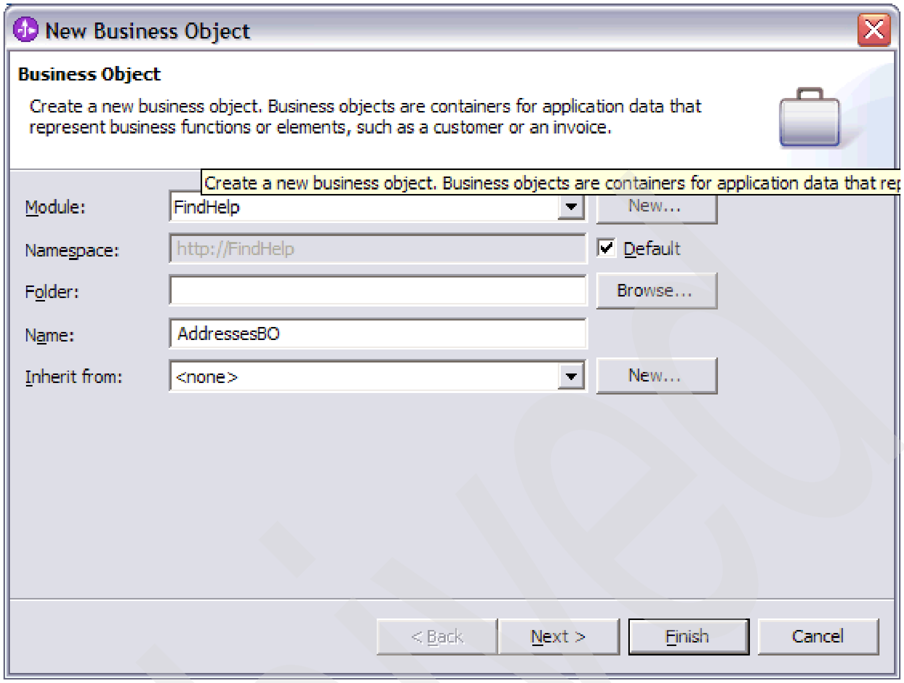


Figure 12-16 New business object wizard

After the new business object is created, it should appear in the business integration view, and should open in the business object editor.

7. Add attributes by performing the following:
   a. Select the business object AddressesBO.
   b. Click the **Add attribute** icon in the business object editor action bar.

> **Note:** Alternatively, you can right-click the business object to open the context menu and then select **Add attribute.**

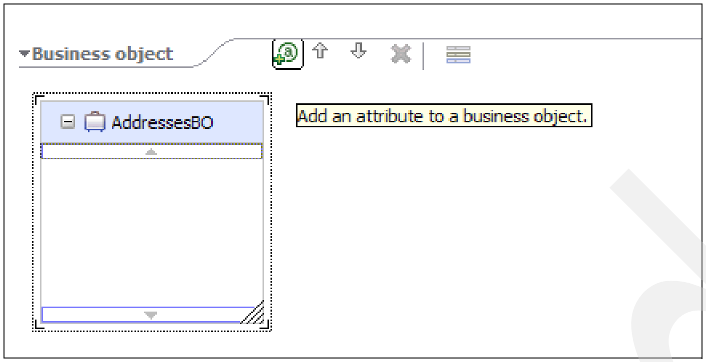


Figure 12-17 Add attribute action button

8. Change the attribute1 name to `address`

> **Note:** The name of the attribute can be changed from either the properties view or the business object editor.

The AddressesBO business object represents an array of addresses.

9. Switch to the properties view.
10. In the properties view of the attribute check **Array** (see Figure 12-18 on page 368).

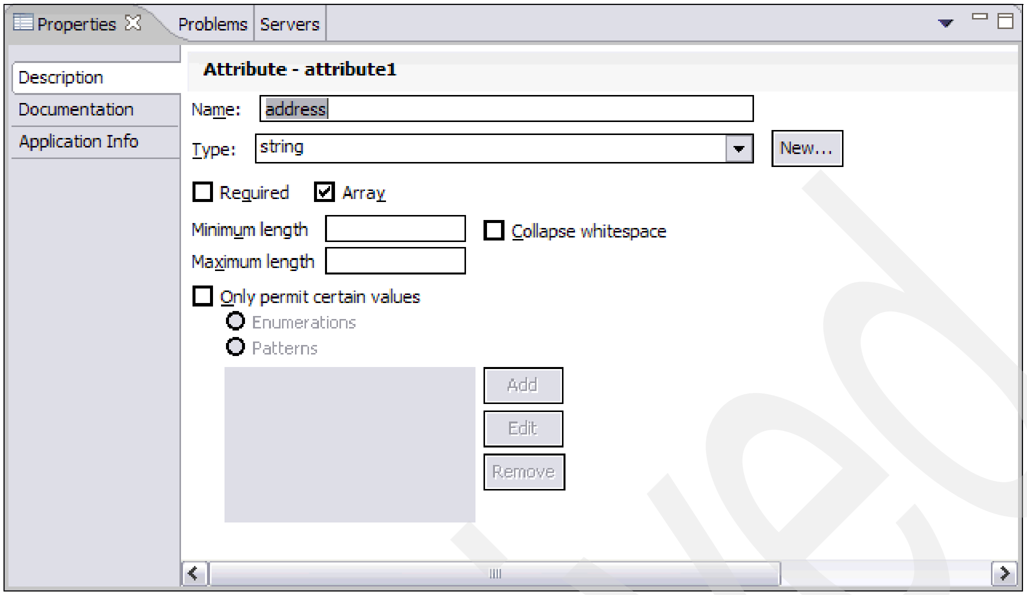


Figure 12-18 The attribute properties view

11. Save and close the business object editor.

12.3.3 Create the interface for the BPEL process

In this section we describe the creation of a new interface for the BPEL process. The interface offers a single operation and a communication channel between the SIP Servlet (see 12.2, "SIP Servlet development" on page 342) and the BPEL process. This interface will use the AddressesBO business object we created in the previous section.

To create the interface, perform the following steps:

1. Right-click the folder **Interfaces** and select **New** → **Interface**.

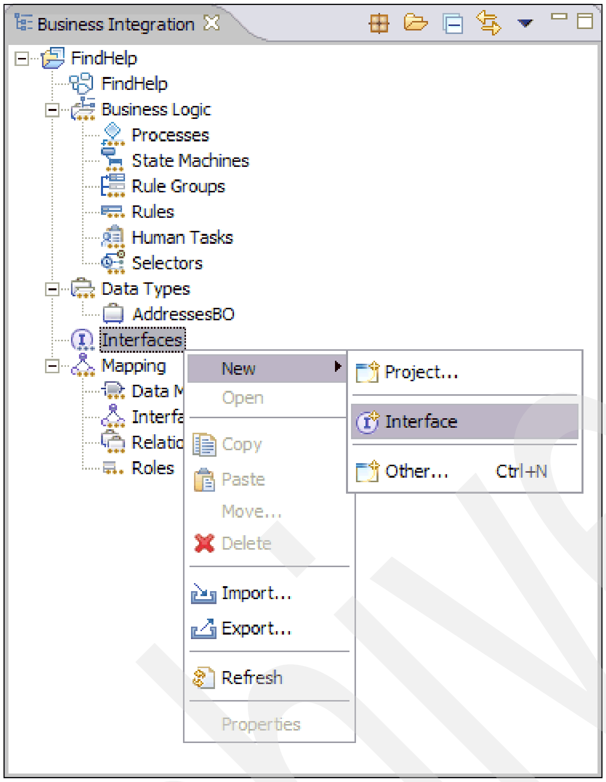


Figure 12-19 Select new interface

2. The New Interface Wizard will be displayed as in Figure 12-20 on page 370.
3. Enter the interface and module information.
   a. Enter `FindHelpInterface` as the interface Name.
   b. Verify that the selected module is `FindHelp`.
4. Click **Finish**.

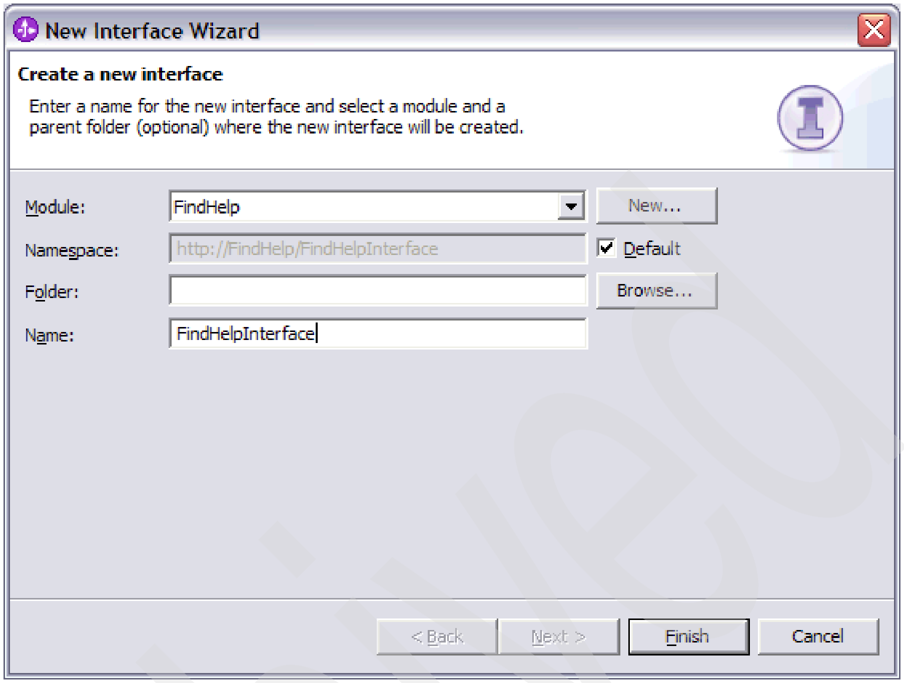


Figure 12-20 New interface wizard

5. The wizard will create the new interface and open it in the interface editor.

We can now create the operation and the parameters.

6. Click the **Add Request Response Operation** icon  in the interface editor.

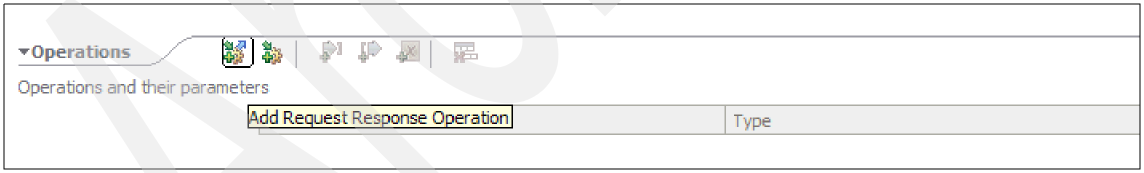


Figure 12-21 Add Request Response Operation action button

7. Change the operation name to `invokeCallBack`.

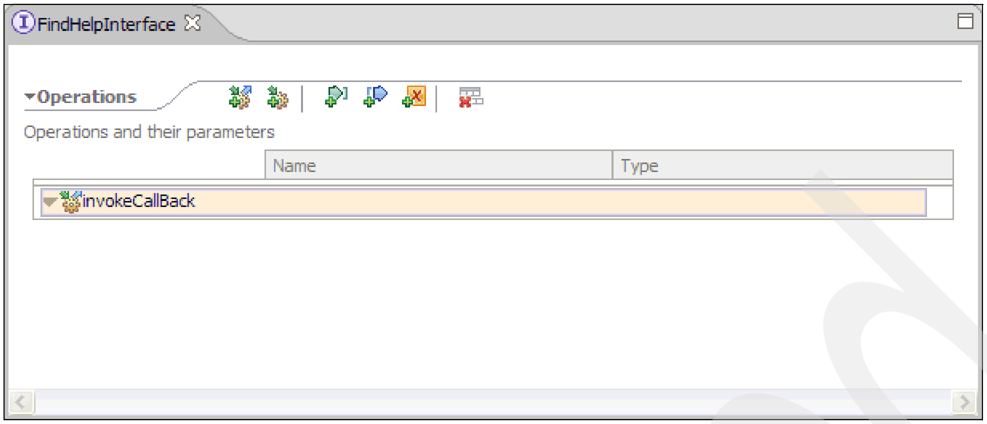


Figure 12-22 Add a new request response operation

We need to add the input parameters `addresses and originator for the invokeCallBack interface.`

8. Click the **Add Input** icon to specify `addresses` input.
9. Change the input1 name to `addresses`.

> **Note:** The data type for `addresses` parameter is the AddressesBO business object we created in 12.3.2, "Create the business object" on page 364.

10. Select the addresses parameter and switch to the Properties view.
11. This opens the Data Type Selection window (see Figure 12-23). Select `Business Objects` as the Parameter Type.

Figure 12-23 Select parameter type

12. In the Data Type Selection dialog (Figure 12-24 on page 372), select the `AddressesBO` business object.

13. Click **OK**

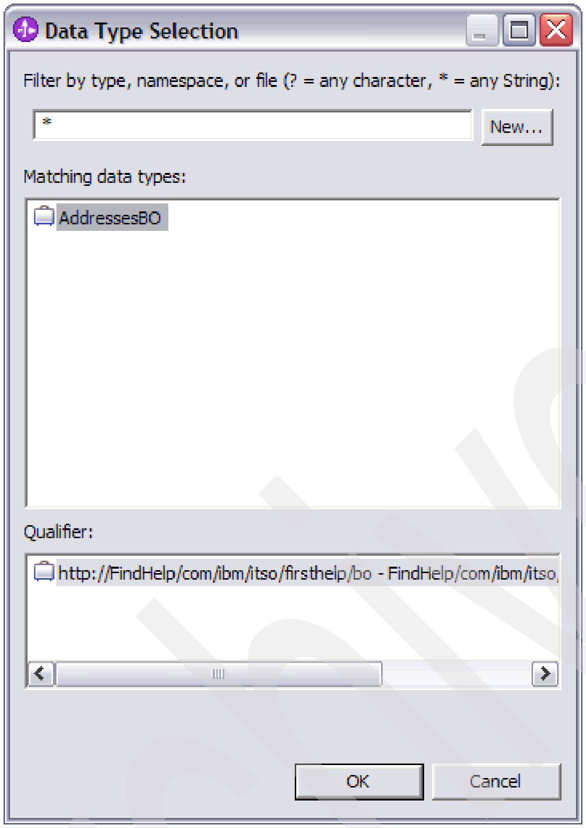


Figure 12-24 Data type selection

14. Click the Add Input icon to specify `originator` input.
15. Change the name from input1 to `originator`.
16. Accept `string` as type for the input parameter.

`The invokeCallBack interface` has two output parameters `status and callee`.

17. Add `status` output parameter.
18. Click the Add Output icon .
19. Change the name from output1 to `status`.
20. Accept `string` as the data type.
21. Add `callee` output parameter.
22. Click the Add Output icon .

23. Change the output1 name to `callee`.

24. Accept `string` as the data type.

The completed interface is shown in Figure 12-25.

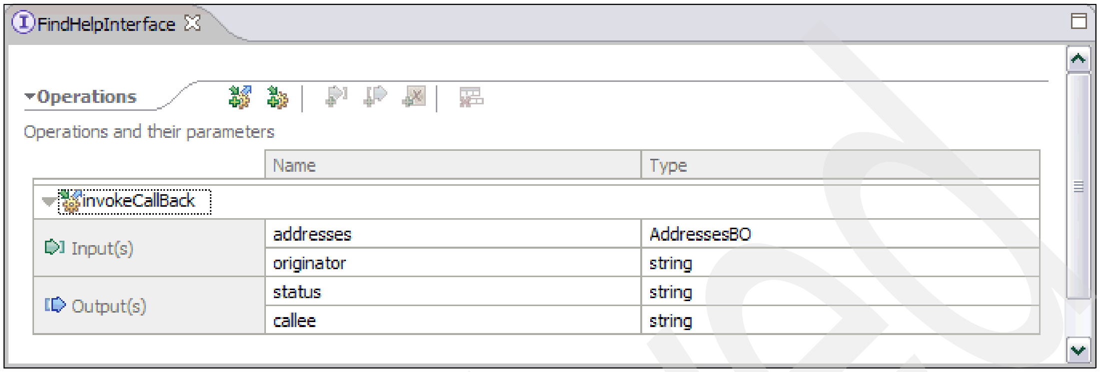


Figure 12-25 Completed FindHelpInterface interface

25. Save and close the interface editor.

12.3.4 Import the WSDL files

The interfaces for the Location, Third Party Call and the Diameter Offline Charging services are defined using WSDL files. In this section we describe the steps you need to go through to import the WSDL files into your module.

Import Location service WSDL files

You need to have installed the Location service, which is provided in the additional material to this redbook.

> **Note:** For information about how to install and setup the Location service, refer to 13.2.2, “Location server setup” on page 448, and also see Appendix C, “Additional material” on page 637.

Perform the following steps to acquire and then import the Location service WSDL file into your module:

1. Logon to the administration console of the application server on which you have deployed the location services. Point your browser to: `http://localhost:9060/ibm/console`.

> **Note:** You may need to change server and port in the above URL to reflect setup for your installation.

2. In the task view click the **Applications** folder to expand it.
3. Click **Enterprise Applications** to get a list of the applications that are deployed on this server (see Figure 12-26).

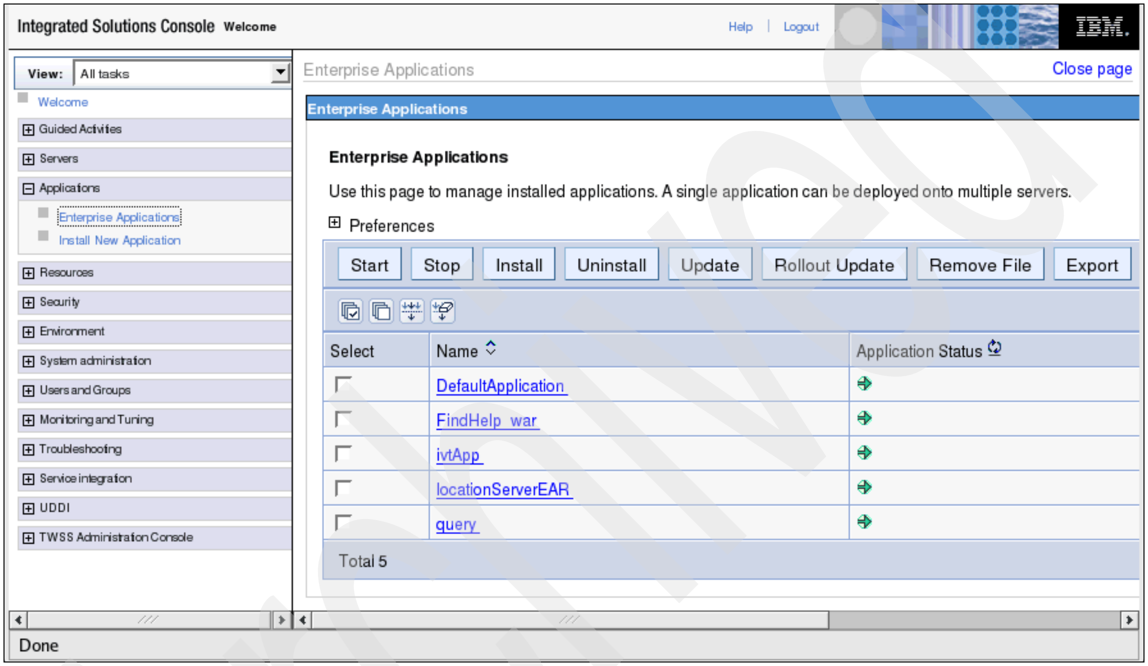


Figure 12-26 Deployed enterprise applications

4. In the list of deployed enterprise applications click **locationServerEAR**. This will open the Location server configuration.

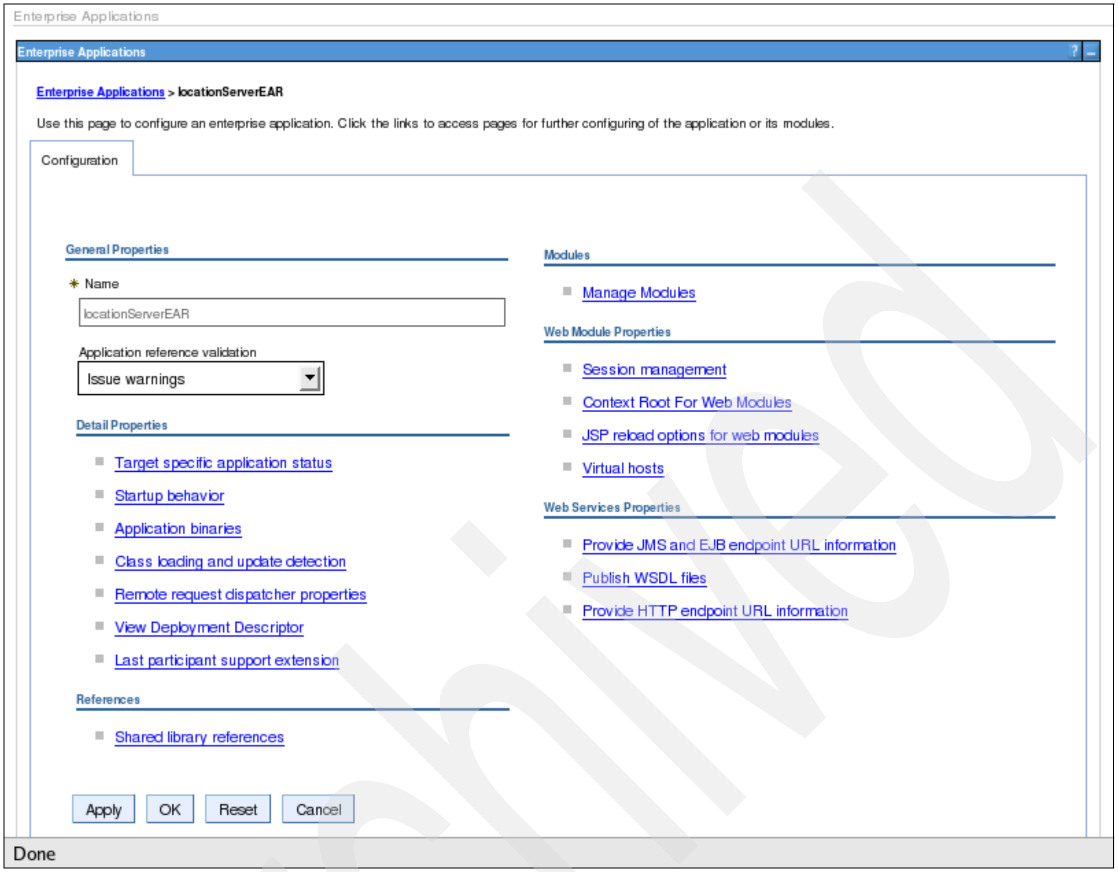


Figure 12-27 The location server configuration

5. In the Web Services Properties section, click **Publish WSDL files**.

> **Note:** The WSDL files are packaged in archive files.

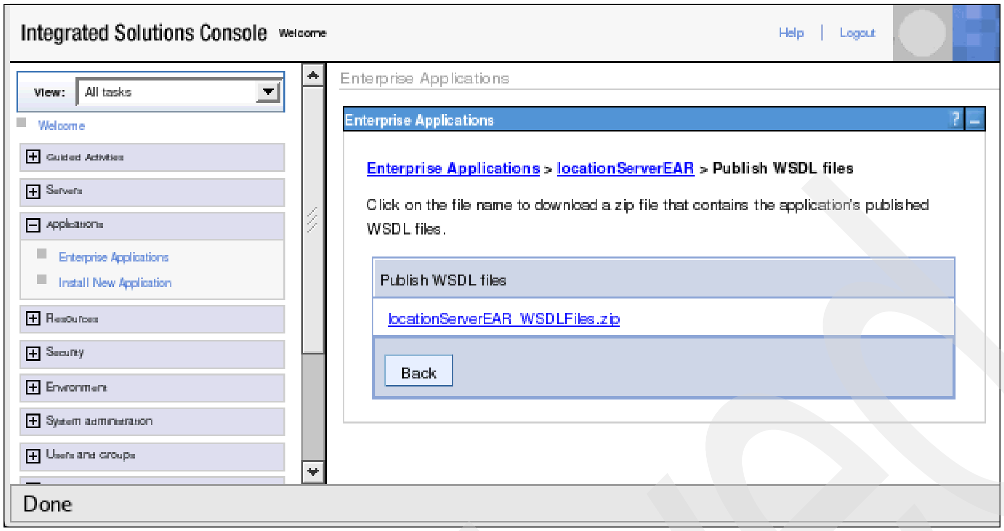


Figure 12-28 The published archive

6. Click the filename **locationServerEAR_WSDLFile.zip** to start the download.
7. Save the archive in a temporary directory.
8. Unpack the archive to the temporary directory.
9. Switch back to the WebSphere Integration Developer.
10. Right-click the FindHelp module and select **Import**.

    This will open the Import wizard (see Figure 12-29 on page 377).
11. Select **Interface/WSDL File**.
12. Click **Next**.

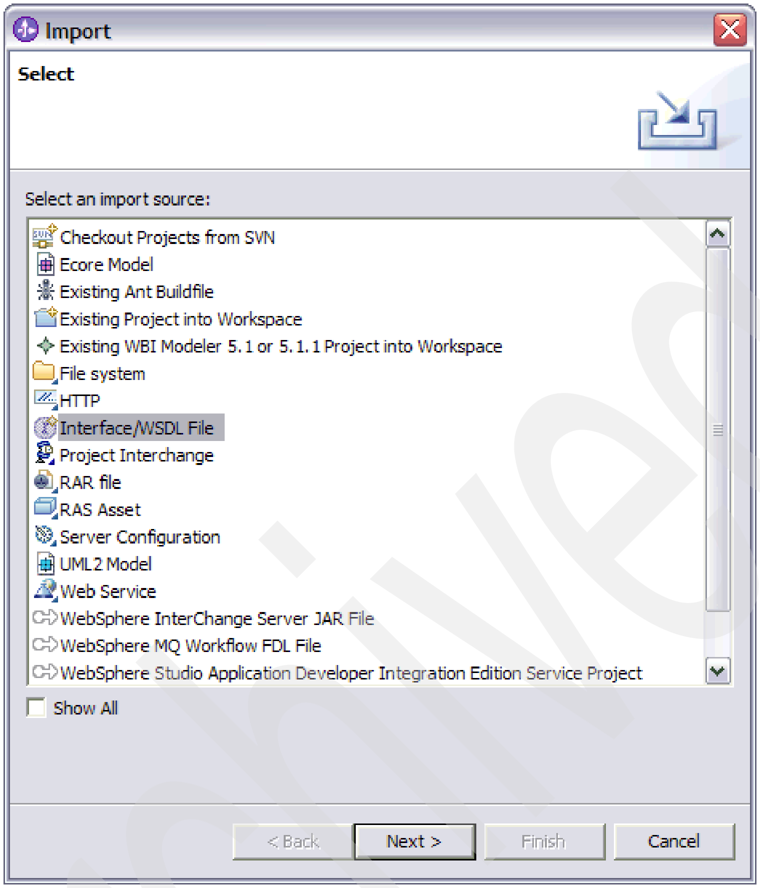


Figure 12-29 The file import wizard

13. Click **Browse** and navigate to the TerminalLocationImpl.wsdl file.

    This file will reside in the temporary directory (see "Unpack the archive to the temporary directory." on page 376) in the subdirectory: `\LocationServer\WebContent\WEB-INF\wsdl\`.

14. Select **TerminalLocationImpl.wsdl**.
15. Enter `FindHelp` as the Into folder.
16. Click **Finish** to start the import.

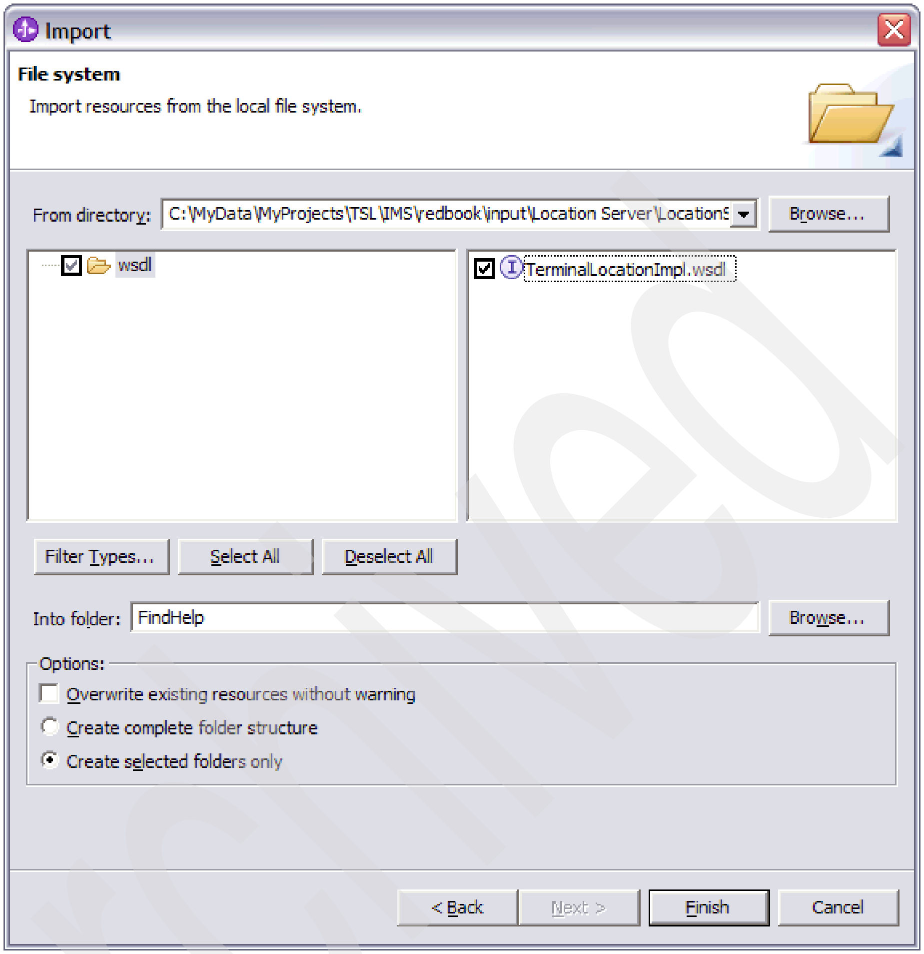


Figure 12-30 Import file selection

17. When the importation is completed, you will see the following new members in the Data Types, Interfaces and Web Service Ports folders of the FindHelp module.

Table 12-2 Location service interface artifacts

Folder	Membername
Interfaces	TerminalLocationImpl
Web Service Ports	TerminalLocationImpl

Folder	Membername
Data Types	ArrayOf_tns3_nillable_LocationData
	ArrayOf_xsd_nillable_anyURI
	ArrayOf_xsd_nillable_string
	LocationData
	LocationInfo
	PolicyException
	RetrievalStatus
	ServiceError
	ServiceException

Import Third Party Call Control service WSDL files

Before you initiate the importation of the Third Party Call Control service WSDL files, you need to have installed the WebSphere Telecom Web Services Server.

> **Note:** See B.2, “IBM WebSphere Telecom Web Services Server” on page 540 for a description of the steps for installing the IBM WebSphere Telecom Web Services Server.

Import the Third Party Call Control service WSDL files following the same steps as in “Import Location service WSDL files” on page 373. Use the following values for the importation:

- `IMS Third Party Call` for the application name
- `IMS Third Party Call_WSDLFiles.zip` for the WSDL archive file name
- `\IMS Third Party Call.ear\thirdparty-web.war\WEB-INF\wsdl` for the subdirectory that will contain the WSDL files
- Select all files to import:
  - **px_cmn_f_2_0.wsdl**
  - **px_cmn_t_2_1.xsd**
  - **px_tpc_i_2_1.wsdl**
  - **px_tpc_s_2_1.wsdl**
  - **px_tpc_t_2_1.xsd**

- When the importation is completed, you will see the following new members in the Data Types, Interfaces and Web Service Ports folders of the FindHelp module.

Table 12-3 Third Party Call Control interface artifacts

Folder	Membername
Interfaces	ThirdPartyCall
Web Service Ports	ThirdPartyCall
Data Types	CallInformation
	CallStatus
	CallTerminationCause
	cancelCallRequest
	cancelCallRequestResponse
	ChargingInformation
	endCall
	endCallResponse
	getCallInformation
	getCallInformationResponse
	makeCall
	makeCallResponse
	PolicyException
	ServiceError
	ServiceException
	SimpleReference
	TimeMetric
	TimeMetrics

Import Diameter Offline Charging service WSDL files

Before you initiate the importation of the Diameter Offline Charging service WSDL files, you need to have installed the WebSphere Diameter Enabler component.

Note: See B.7, “IBM WebSphere Diameter Enabler component” on page 625 for a description of the steps for installing the WebSphere Diameter Enabler component.

Import the Diameter Offline Charging service WSDL files following the same steps as in “Import Location service WSDL files” on page 373. Use the following values for the importation:

- `DHADiameterRfWebServiceEAR` for the application name
- `DHADiameterRfWebServicesEAR_WSDLFiles.zip` for the WSDL archive file name
- `\DHADiameterRfWebServiceEAR.ear\DHADiameterRfWebService.war\WEB-INF\wsdl` for the subdirectory that will contain the WSDL files
- Import the WSDL file named `DiameterRfService.wsdl`
- When the importation is completed, you will see the following new members in the Data Types, Interfaces and Web Service Ports folders of the FindHelp module.

Table 12-4 Third Party Call Control interface artifacts

Folder	Membername
Interfaces	DiameterRfService_SEI
Web Service Ports	DiameterRfService

Folder	Membername
Data Types	ACAResults
	Accounting
	AppServInfo
	ArrayOf_tns2_nillable_SDPmedia
	ArrayOf_tns3_nillable_Avp
	ArrayOf_tns3_nillable_VsAvp
	ArrayOf_xsd_int
	ArrayOf_xsd_string
	Avp
	AvpValueUtil
	AvpValueUtilGrouped
	AvpValueUtilOctetString
	AvpValueUtilUnknown
	AvpValueUtilUnsigned32
	AvpValueUtilUnsigned64
	AvpValueUtilUTF8String
	CauseCode
	SDPmedia
	SipInfo
	TrunkGroup
	UUSdata
	Vector
	VsAvp

Important: Your imported WSDL may contain a reference to the WS-Addressing service. This will cause errors when you run the service. Therefore, you need to correct the WSDL and remove the following line.

1. In the Business Integration view expand the Interfaces folder.
2. Select the DiameterRfService_SEI interface.
3. Right-click and select **Open With** → **XML Source Page Editor**.
4. This opens the DiameterRfService.wsdl.
5. In the file search for **"**`wsaw`**"**.
6. If your WSDL file contains a line similar to: `<wsaw:UsingAddressing wsdl:required="false" xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"/>`

   Then, remove the complete line and save your WSDL file.

12.3.5 Create the business process

In this section we will create the FindHelp business process. To create a new business process execute the following steps:

1. Go to the Business Integration view.
2. Expand the folder Business Logic.
3. Right-click the folder Processes.
4. Select **New** → **Business Process** (see Figure 12-31 on page 384).
5. In the New Business Process window, enter `FindHelpBP` as the Name for the business process.
6. Click **Next**.

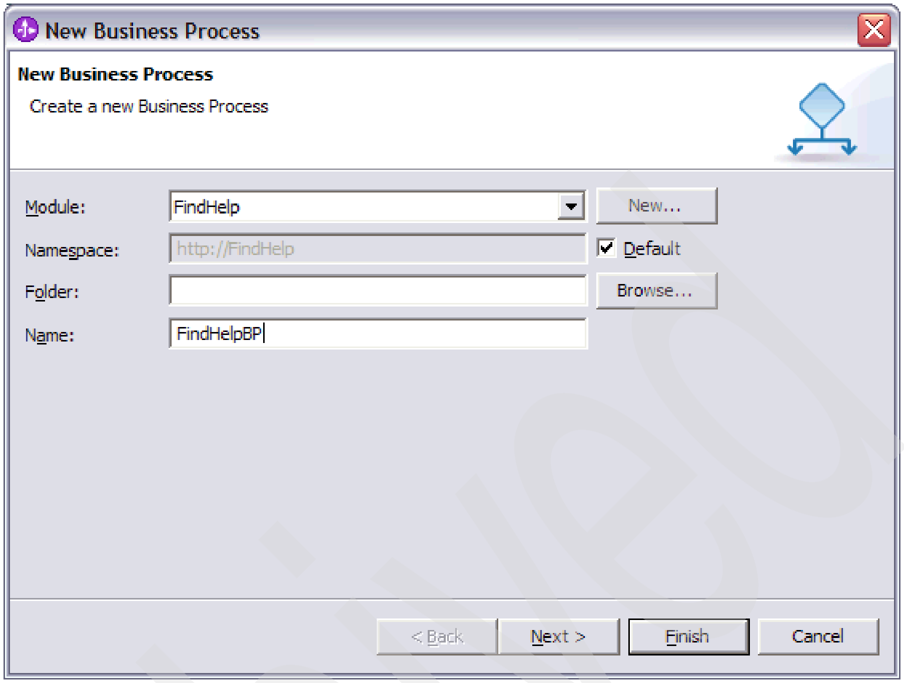


Figure 12-31 The new business process wizard

> **Note:** You should have defined the interface beforehand as we did in 12.3.3, "Create the interface for the BPEL process" on page 368, or you can define the interface here before you proceed.

7. Select the option **Select an existing Interface**.
8. Click **Browse**.
9. From the Interface Selection dialog box (see Figure 12-32 on page 385), select the **FindHelpInterface**.
10. Click **OK**.

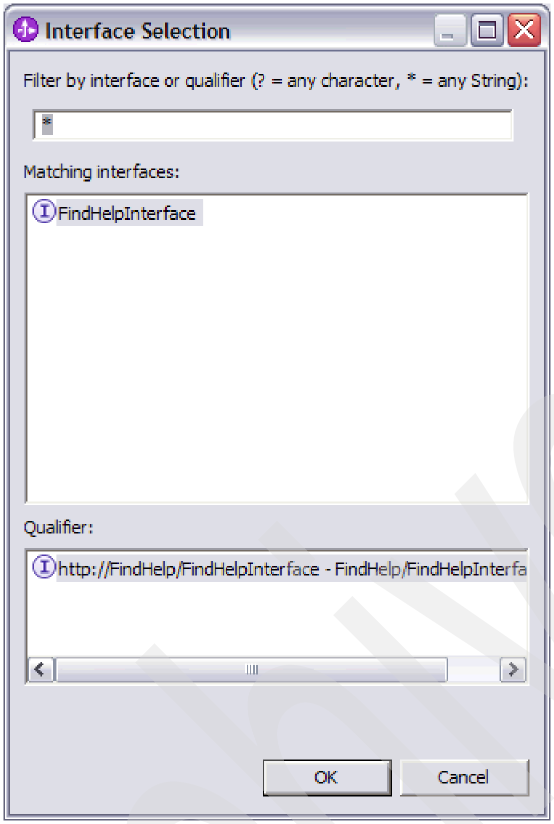


Figure 12-32 Interface selection

11. The selected interface is now visible in the window shown in Figure 12-33 on page 386.
12. Click **Finish**.

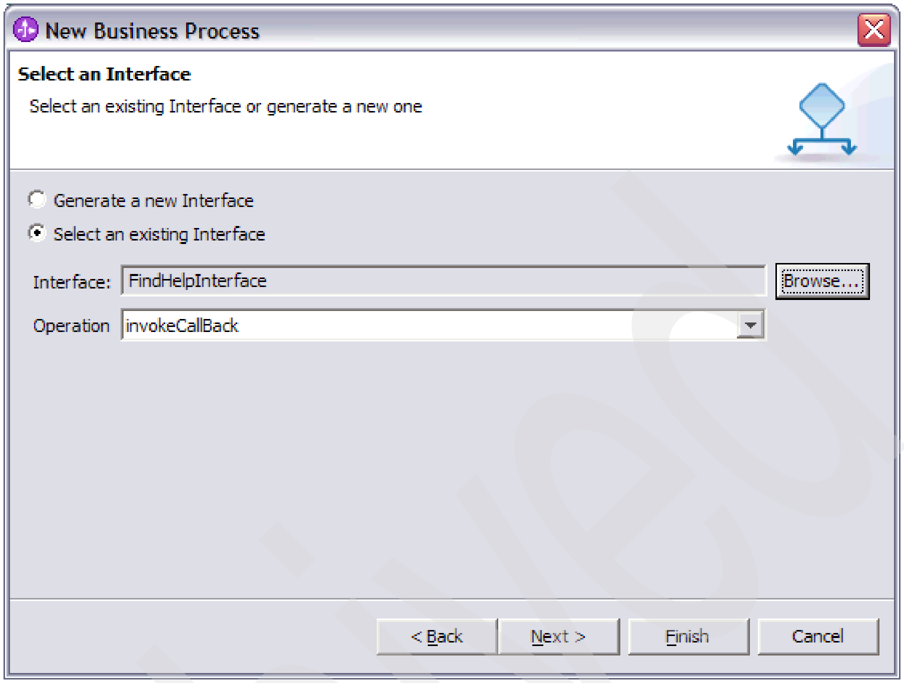


Figure 12-33 Select an Interface

The new process FindHelpBP will be generated and displayed in the Business Process editor. Notice that a Receive node the entry point into the business process was added. The generated Reply node returns the output parameters to the component that invoked the interface.

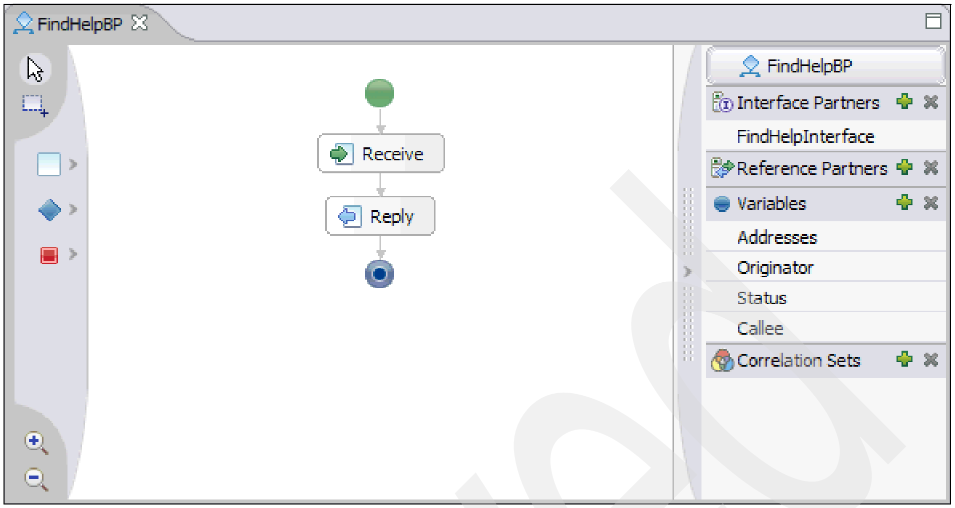


Figure 12-34 The FindHelpBP business process editor

12.3.6 Add partner references

The BPEL specification refers to external services that interact with the process as partners. We need to create Reference Partners for the external services whose WSDL files we imported in 12.3.4, “Import the WSDL files” on page 373.

1. In the Business Integration view, navigate to the **FindHelpBP** in the Processes folder.
2. Double-click **FindHelpBP** to open it in the Business Process Editor.
3. In the business process editor’s tray right-click **Reference Partners**.
4. Select **Add Reference Partner** (or click the Add icon right next to the Reference Partner section as shown in Figure 12-35 on page 388).

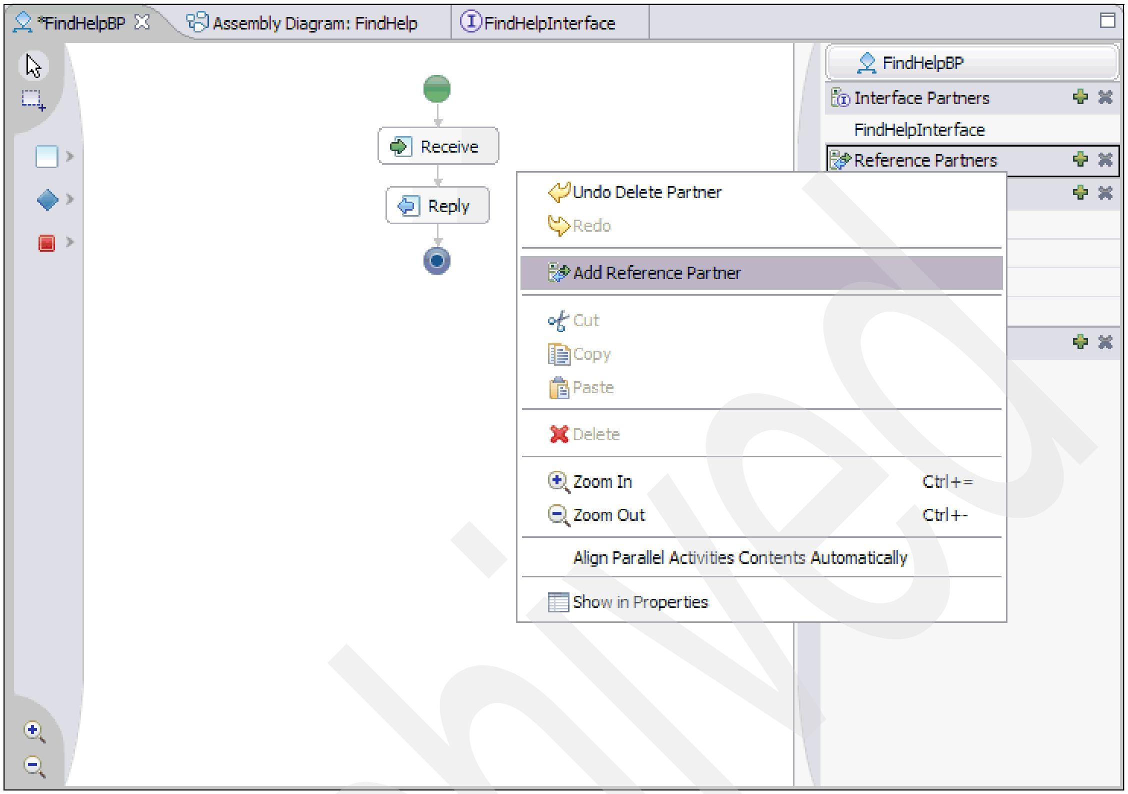


Figure 12-35 Add a new reference partner

5. Change the Partner name to `LocationServicePartner`.

> **Note:** You can also change the name of the partner reference in the Properties view.

6. In the Properties view, select the **Details** tab.
7. Click **Browse**. This will open the interface selection window (see Figure 12-36 on page 389).

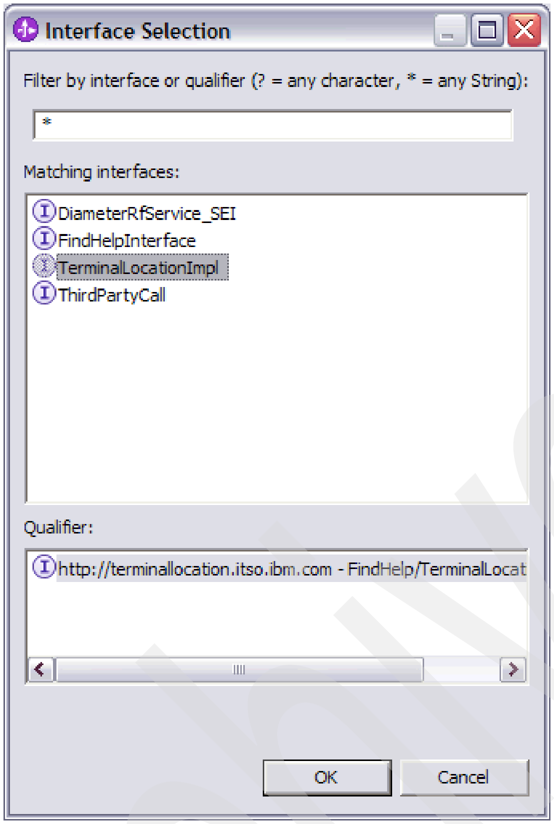


Figure 12-36 Select the interface for the reference partner

8. Select the **TerminalLocationImpl** interface.
9. Click **OK**.

When finished, the Details tab will look similar to Figure 12-37.

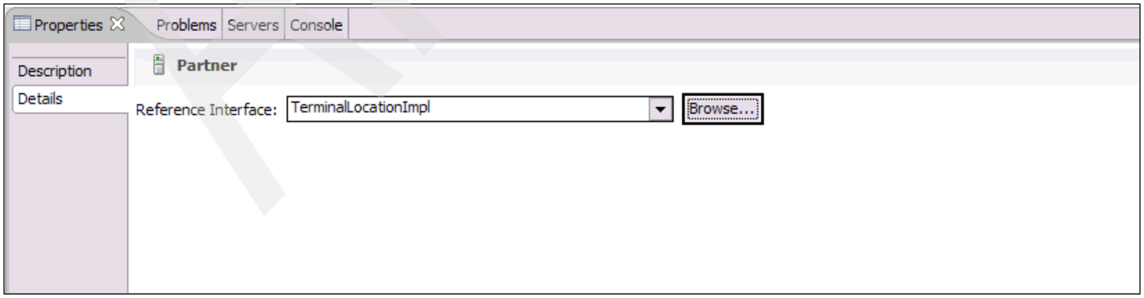


Figure 12-37 Reference partner details

10. Repeat Steps 3 to 9 to create reference partners for the remaining two services using the following values:

Table 12-5 Reference partner and interface names

Reference Partner Name	Interface
ThirdPartyCallControlPartner	ThirdPartyCall
DiameterRfPartner	DiameterRfService_SEI

11. Save and close the business process editor.

12.3.7 Add process logic

This section focuses on the implementation of the process logic for the FindHelpBP business process, which was created in 12.3.5, "Create the business process" on page 383.

The FindHelp process is a strict sequential process. Figure 12-39 on page 392 shows the process outline. The main process interactions includes the following building blocks:

- Init Process - initializes the process
- Process Addresslist - processes the addresslist to determine the closest address
- Return Reply - returns a response to the invoking client
- Establish Call - establishes a call between originator and the closest address
- Create Charging Event - creates a charging event

To build the process outline follow these steps:

1. In the Business Integration view, expand the Business Logic and Processes folder.
2. Double-click **FindHelpBP** to open it in the Business Process editor.
3. Select the **Receive** node.
4. Right-click and select **Insert before** → **Sequence**.

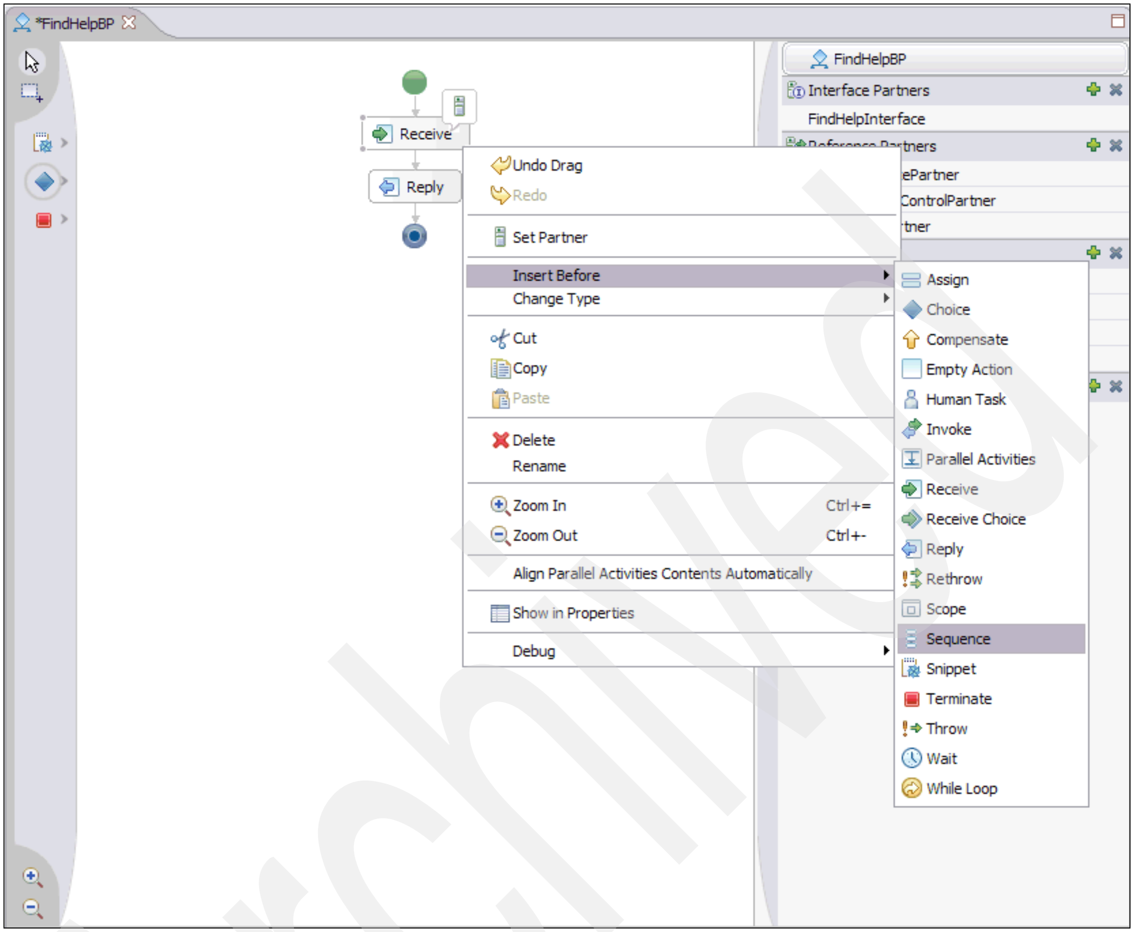


Figure 12-38 Insert a sequence

5. Change the default name to `Init Process`.
6. Drag the Receive node and drop it inside the Init Process sequence.
7. Select the Reply node.
8. Right-click and select **Insert before** → **Choice**.
9. Change the default name to `Process Addresslist`.
10. Select the Reply node.
11. Right-click and select **Insert before** → **Sequence**.
12. Change the default name to `Return Reply`.
13. Drag the Reply node and drop it inside the Return Reply sequence.

14. Right-click anywhere on the canvas and select **Add** → **Sequence**.
15. Change the default name to `Establish Call`.
16. Right-click anywhere on the canvas and select **Add** → **Sequence**.
17. Change the default name to `Create Charging Event`.
18. Save the business process.

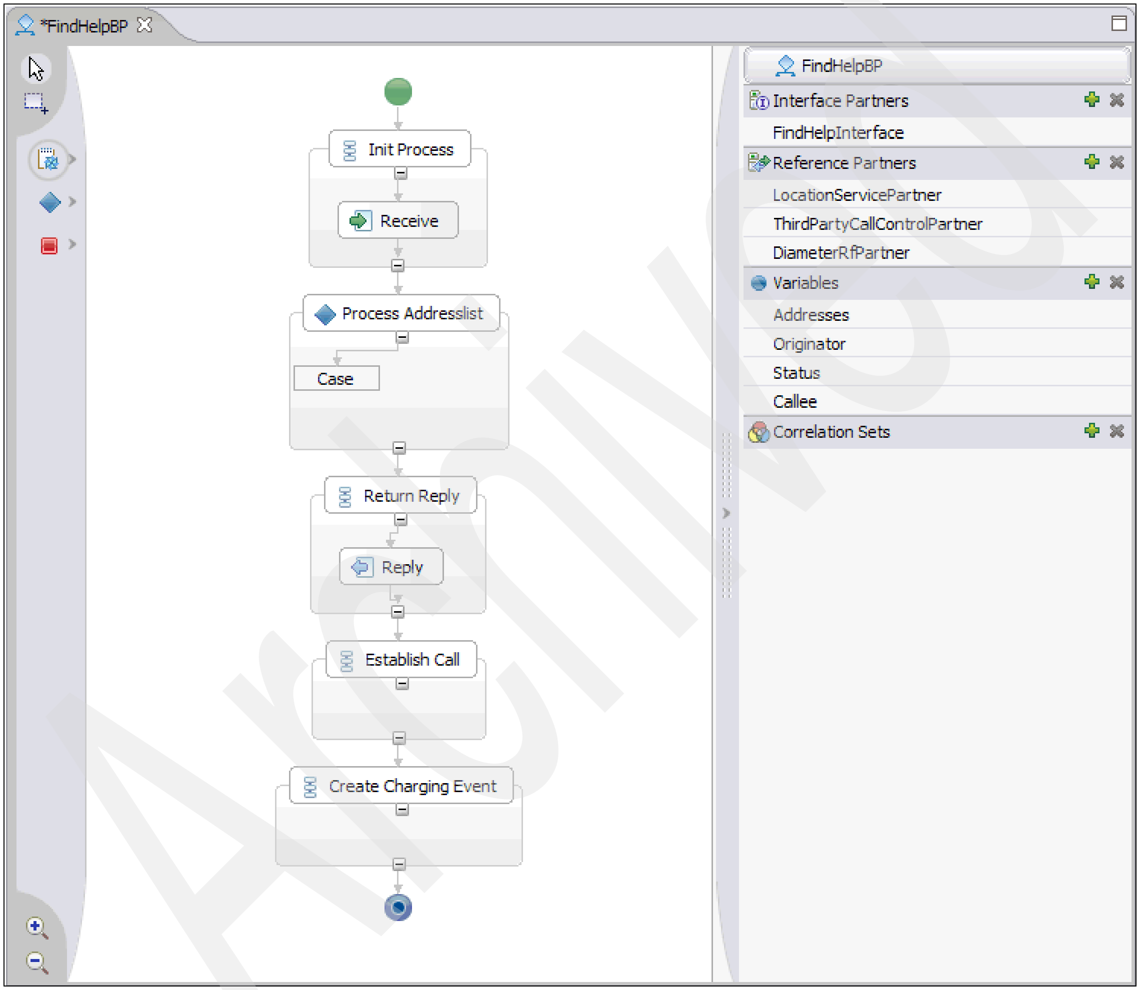


Figure 12-39 Business process outline

With the outline defined, the next step is to walk through each of the process blocks and create the processing logic.

Implement the Init Process sequence

In this sequence we log the receipt of the initial message and initialize process variables.

1. First, we define global variables for the process.
   a. Click the Add icon that is next to the label Variables (see Figure 12-39 on page 392).
   b. Add a new variable called `nNumAddresses`.

      This variable stores the number of addresses in the list, which is passed as input parameter when the process is invoked.
2. Select the `nNumAddresses` variable and open the Properties view.
3. In the Details section, click **Browse** and set the data type to `int`.
4. Add the second variable `nIterator` of data type `int` by repeating steps 1 to 3 above.
5. Add the third variable `bMakeCallfault` of data type `boolean` by repeating steps 1 to 3 above.
6. Now, add a Java Snippet on the canvas.
   a. Right-click the canvas after the Receive node.
   b. Select **Add** → **Snippet**.

> **Tip:** You can also add nodes to the BPEL process by selecting the appropriate node in the pallet on the left side of the business process editor. You simply drag and drop the selected node into place on the canvas where you want to insert it.

7. Rename the new snippet `Init Variables`.
8. Select the snippet and open the Properties view.
9. Select the Details tab.

   You now have access to the visual snippet editor that can be used to write or visually compose the necessary Java code.

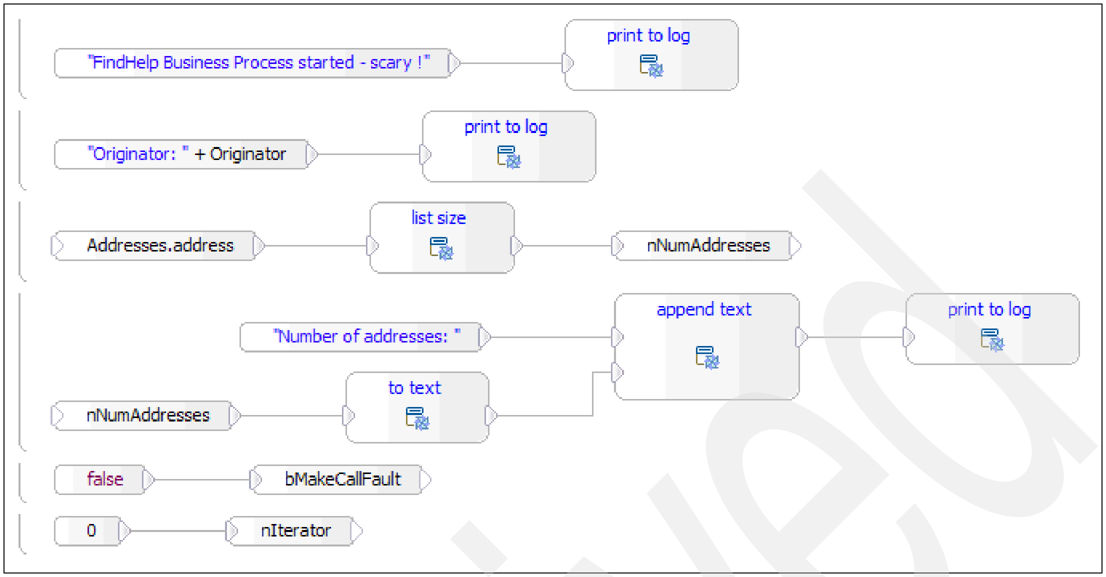


Figure 12-40 Init Variables snippet

In the above snippet we log the start of the process and initialize some internal global variables.

10.Save the business process editor.

Implement the Process Addresslist choice

Three cases are implemented in the choice node:

- **Empty list**: Return error status and terminate the process
- **Single entry in list**: No need to determine the closest address, continue and establish the call
- **Many entries in list**: Start to determine the closest address

Add these cases to the choice node by executing the following steps:

1. Select the default case.
2. Open the Properties view and select the Description tab.
3. Enter `Empty` as Display Name.
4. Select the Details tab.
5. Click **Create a New Condition**. This will open the visual editor.
6. Create a condition as shown in Figure 12-41.

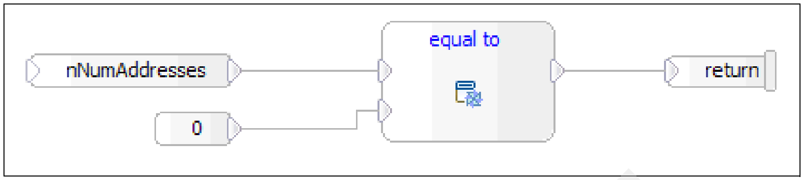


Figure 12-41 Empty condition

> **Note:** This branch is only entered, if the list with addresses is empty, i.e. the number of addresses in the list equals zero.

7. In the business process editor, select the Empty node.
8. Right-click and select **Add** → **Snippet**.
9. Rename the snippet `Assign EmptyList Status`.

   The Process Addresslist node should now look like the one in.Figure 12-42.

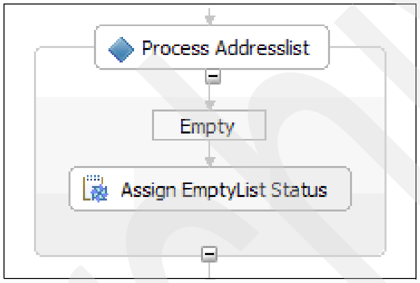


Figure 12-42 Process Addresslist with Empty Case

10. With the snippet selected, open the Properties view.
11. Select the Details tab.
12. Create a visual snippet as shown in Figure 12-43 on page 395.

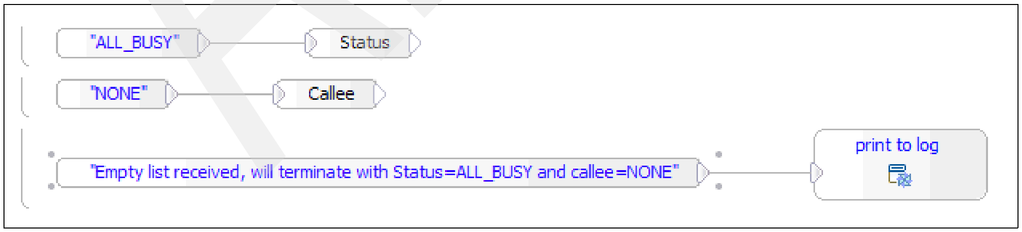


Figure 12-43 Assign EmptyList Status snippet

This snippet sets the status output parameter to “ALL_BUSY” and writes a log.

To add a second case statement to the choice node follow these steps:

1. Select the Process AddressList choice node.
2. Right-click and select **Add Case**.
3. Select the new case.
4. Open the Properties view and select the Description tab.
5. Enter `Single Entry` as Display Name.
6. Select the Details tab.
7. Click **Create a New Condition**. This will open the visual editor.
8. Create a condition as shown in Figure 12-44.

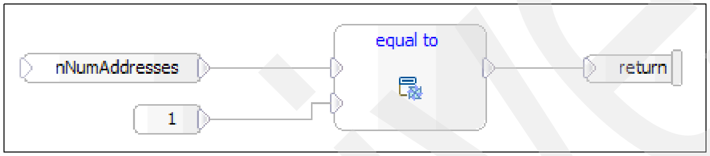


Figure 12-44 Single Entry condition

This branch is only entered, if there is only one address in the list.

9. In the business process editor, select the Single Entry node.
10. Right-click and select **Add** → **Snippet**.
11. Rename the snippet `Assign First Address`.

    The Process Addresslist node should look similar to Figure 12-45 on page 396.

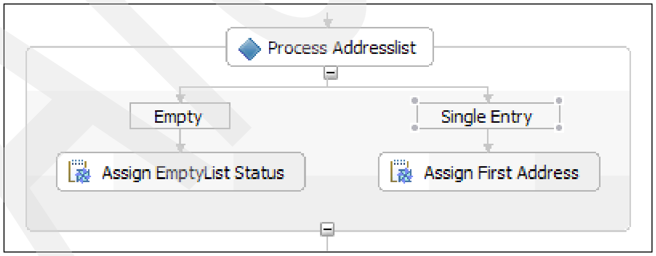


Figure 12-45 Process Addresslist with Single Entry case

12. With the snippet selected, open the Properties view.
13. Select the Details tab.
14. Create a visual snippet as shown in Figure 12-46.

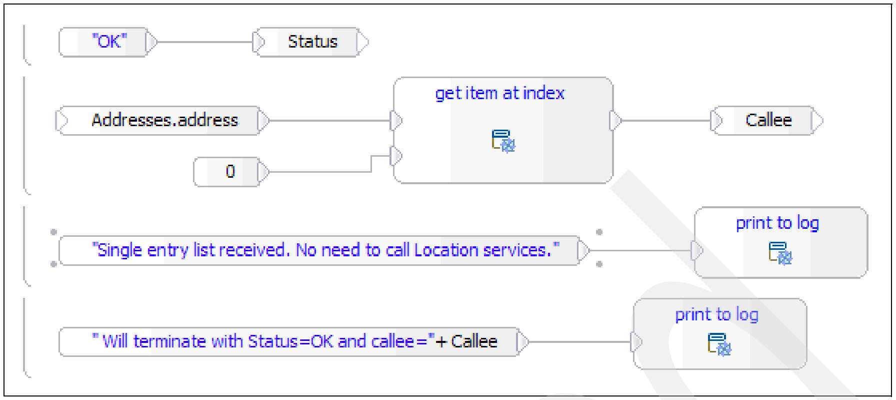


Figure 12-46 Assign First Address snippet

This snippet assigns the one and only address in the list to the output parameter Callee and writes a log.

Next, we add the Otherwise case to the Choice node. This case will catch all other conditions (lists with more than one address). If there are multiple addresses, we need to determine the one that is closest to the originator. To do so, follow these steps:

1. Select the Process AddressList choice node.
2. Right-click and select **Add Otherwise**.
3. Select the new Otherwise case.
4. Right-click and select **Add** → **Sequence**.
5. Rename the new sequence `Determine Closest Address`.
6. The Process Addresslist node should look like Figure 12-47.

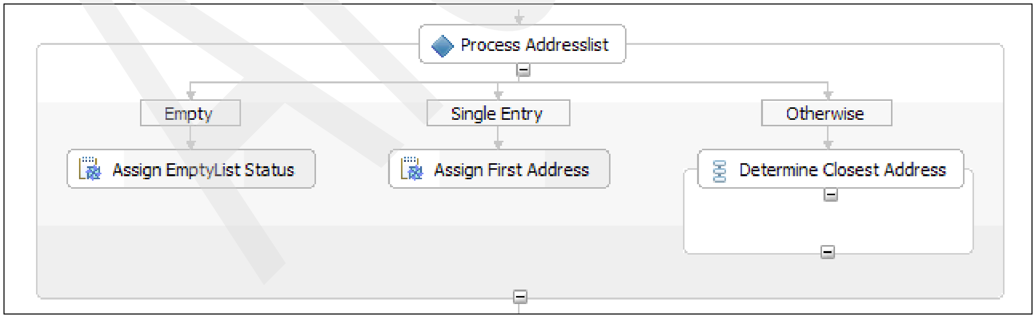


Figure 12-47 Process Addresslist with Otherwise case

Before invoking the Location service we need to define the required input and output variables. Using the Add icon next to the Variables label add the variables as described in Table 12-6.

Table 12-6 Variables for Determine Closest Address sequence

Variable	Data type
uriAddress	anyURI
nRequestedAccuracy	int
nAcceptableAccuracy	int
nMinDistance	int
sMinDistanceAddress	string
fOrigLongitude	float
fOrigLatitude	float
nCurDistance	int
sCurDistanceAddress	string
LocationResponse	LocationInfo

Tip: To assign the anyURI data type you need to check the **Show all XSD types** check box in the Data Type Selection dialog box.

7. Select the Determine Closest Address sequence.
8. Right-click and select **Add** → **Snippet**.
9. Rename the new snippet `Assign GetLocation Input`.
10. With the snippet selected, open the Properties view .
11. Select the Details tab.
12. Create a visual snippet as shown in Figure 12-48.

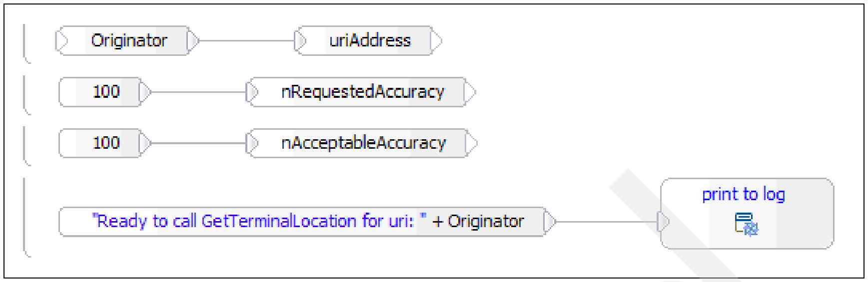


Figure 12-48 Assign GetLocation Input snippet

> **Note:** We use 100 meter as accuracy. If you change the value for accuracy, make sure that your location simulator database is populated accordingly.

Now it is time to invoke the getLocation operation of the Location Web service.

13. Select the Determine Closest Address node.
14. Right-click and select **Add** → **Invoke**.
15. Rename the new invoke node `GetLocation`.
16. With the invoke node selected, open the Properties view.
17. Select the Details tab.
18. Click **Browse** to select a Reference Partner. The partner selection dialog box will be opened.
19. Select **LocationServicePartner**.
20. Click **OK**.

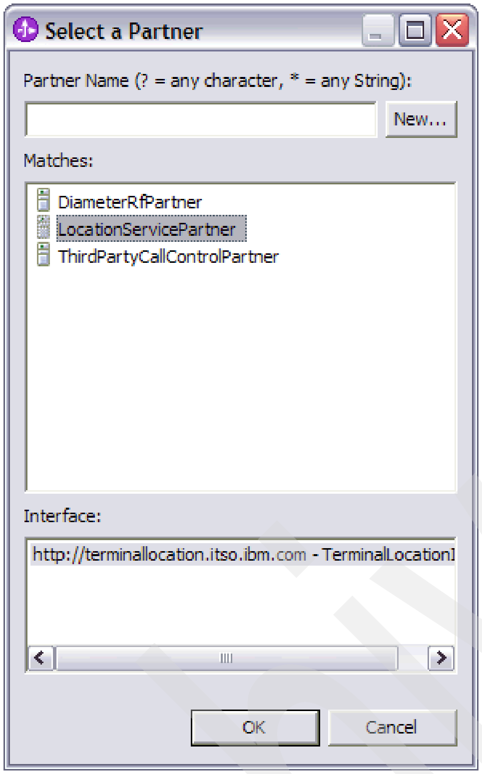


Figure 12-49 Select a reference partner

21. The selected Partner and Interface are displayed in the Details tab.
22. Choose **getLocation** as the operation.
23. Make sure that **Use Data Type Variables** check box is checked.
24. To assign a variable to the URI input parameter, click the ... icon at the end of the line. The Select Variable for URI dialog box opens (see Figure 12-50 on page 401).
25. Select **uriAddress**.
26. Click **OK**.

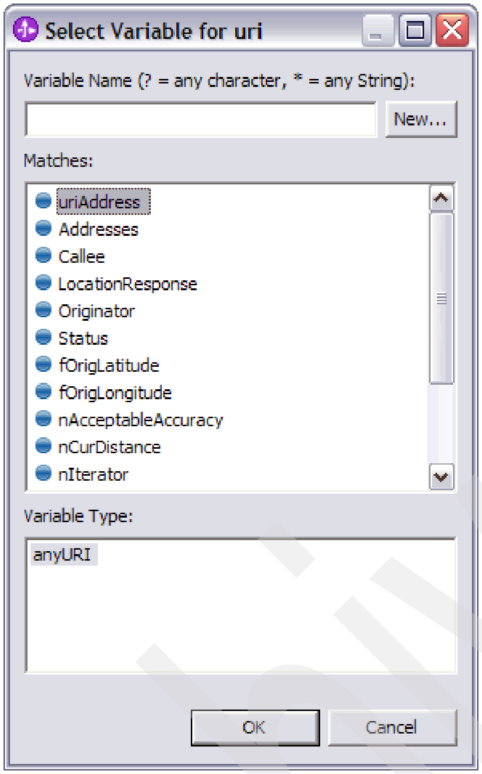


Figure 12-50 Select variable

27. Similarly assign the following variables to the input and output parameters of the getLocation operation:

Table 12-7 Assign variables to the getLocation parameters

Parameter	**Variable**
requestedAccuracy	nRequestedAccuracy
acceptableAccuracy	nAcceptableAccuracy
getLocationReturn	LocationResponse

The completed GetLocation Details tab should look like Figure 12-51 on page 402.

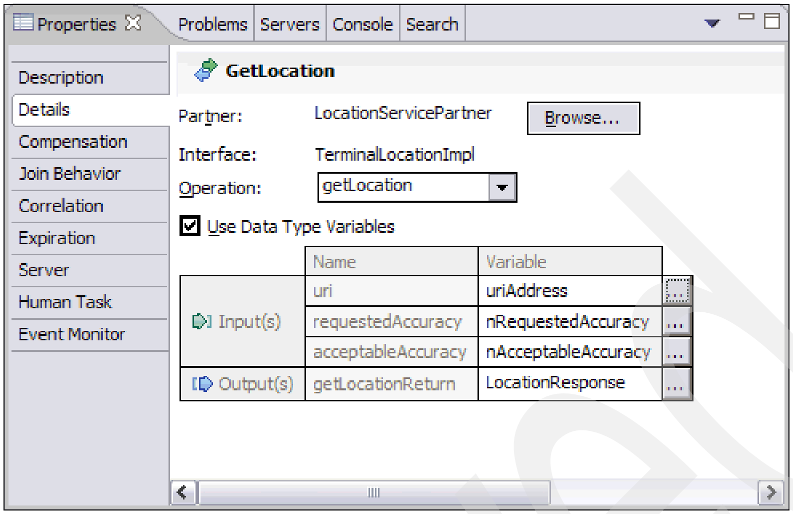


Figure 12-51 GetLocation details

28. Select the Determine Closest Address sequence.
29. Right-click and select **Add** → **Snippet**.
30. Rename the new snippet `Assign GetLocation Output`.
31. With the snippet selected, open the Properties view.
32. Select the Details tab.
33. Create a visual snippet as shown in Figure 12-52.

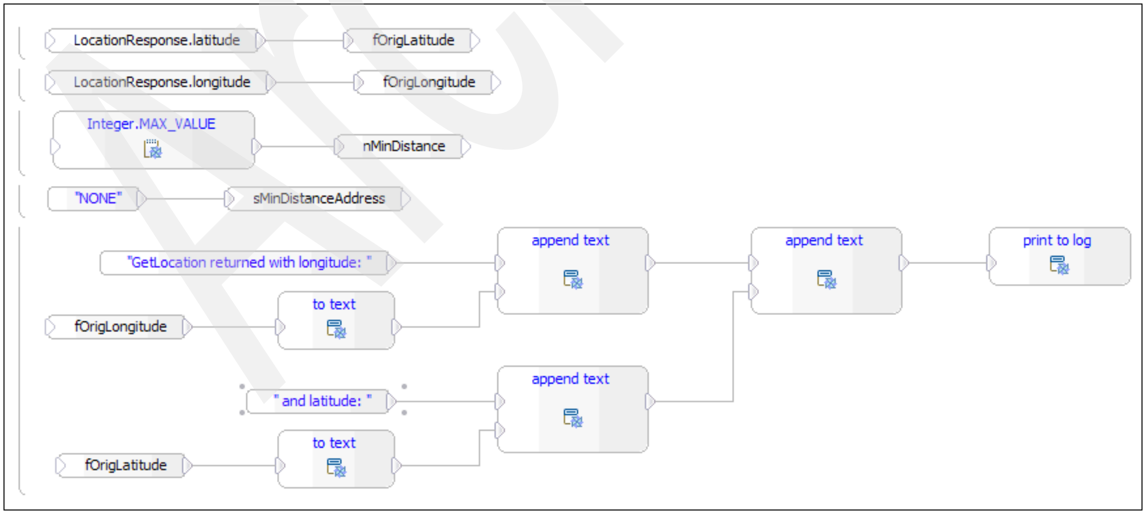


Figure 12-52 Assign GetLocation Output snippet

To calculate the address that is nearest to the originator we need to call the getTerminalDistance for each address in the list:

34. Select the Determine Closest Address sequence.
35. Right-click and select **Add** → **While Loop**.
36. Set the name of the new While loop to `Loop Through Addresslist`.
37. With the snippet selected, open the Properties view.
38. Select the Details tab to specify the loop conditions.
39. In the Details tab select **Create a New Condition**. This opens the visual snippet editor.
40. The loop shall iterate through all entries in the addresslist. The condition is implemented in the snippet as shown in Figure 12-53.

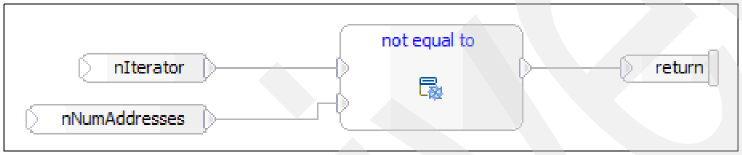


Figure 12-53 Loop through Addresslist condition

41. Select the Loop Through Addresslist While loop.
42. Right-click and select **Add** → **Snippet**.
43. Rename the new snippet `Assign GetDistance Input`.
44. With the snippet selected, open the Properties view.
45. Select the Details tab.
46. Create a visual snippet as shown in Figure 12-54.

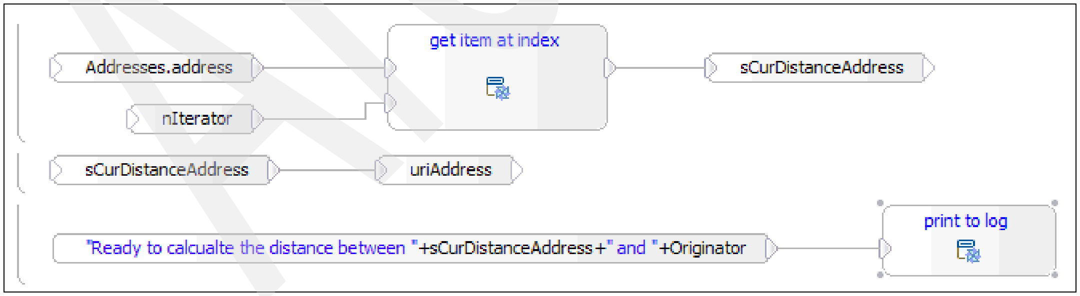


Figure 12-54 Assign GetDistance Input snippet

We are ready to invoke the getTerminalDistance operation of the Location Web service.

47. Select the Loop Through Addresslist While loop.
48. Right-click and select **Add** → **Invoke**.
49. Rename the new invoke node `GetDistance`.
50. With the invoke node selected, open the Properties view.
51. Select the tab Details.
52. Click **Browse** to select a Reference Partner. The partner selection dialog box opens.
53. Select **LocationServicePartner**.
54. Click **OK**.
55. The selected Partner and Interface are displayed in the Details tab.
56. Choose **getTerminalDistance** as the operation.
57. Make sure that **Use Data Type Variables** check box is checked.
58. Similar to the getLocation invoke node (see Step 24 on page 400) assign the following variables to the input and output parameters of the getTerminalDistance operation:

Table 12-8 Assign variables to the getTerminalDistance parameters

Parameter	Variable
uri	uriAddress
latitude	fOrigLatitude
longitude	fOrigLongitude
getTerminalDistanceReturn	nCurDistance

When you are finished, the GetDistance Details tab should look similar to Figure 12-55 on page 405.

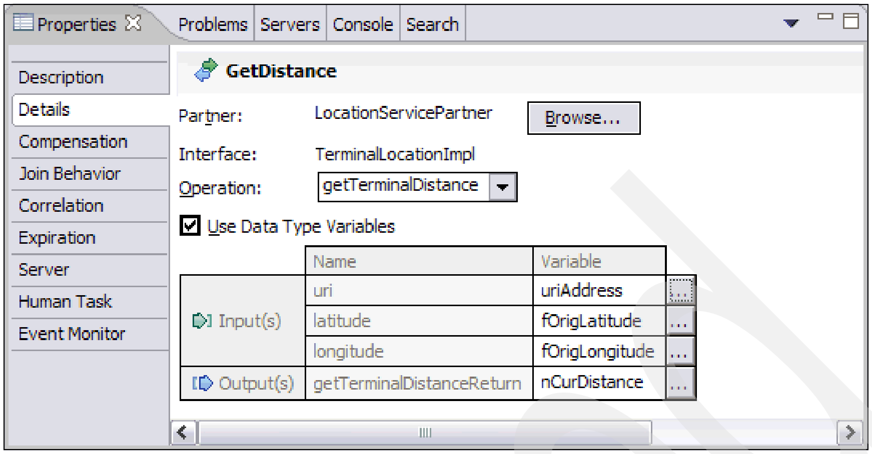


Figure 12-55 GetDistance details

59.Select the Loop Through Addresslist While loop.

60.Right-click and select **Add** → **Snippet**.

61.Rename the new snippet `Assign GetDistance Output`.

62.With the snippet selected, open the Properties view.

63.Select the Details tab.

64.Create a visual snippet as shown in Figure 12-56.

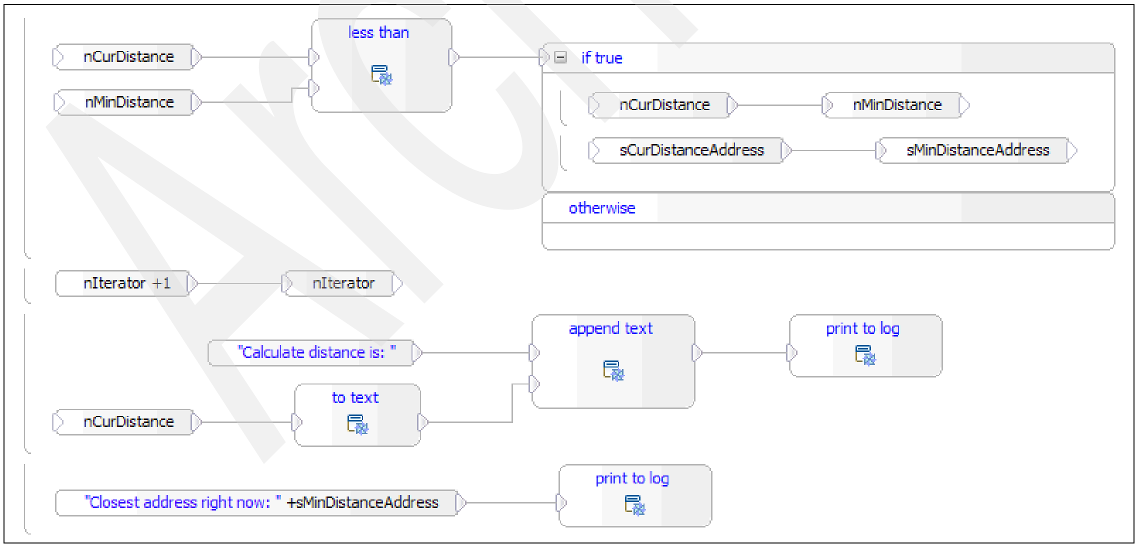


Figure 12-56 Assign GetDistance Output snippet

65. Select the Determine Closest Address sequence.
66. Right-click and select **Add** → **Snippet**.
67. Rename the new snippet `Assign Closest Address`.
68. With the snippet selected, open the Properties view .
69. Select the tab Details.
70. Create a visual snippet as shown in Figure 12-57.

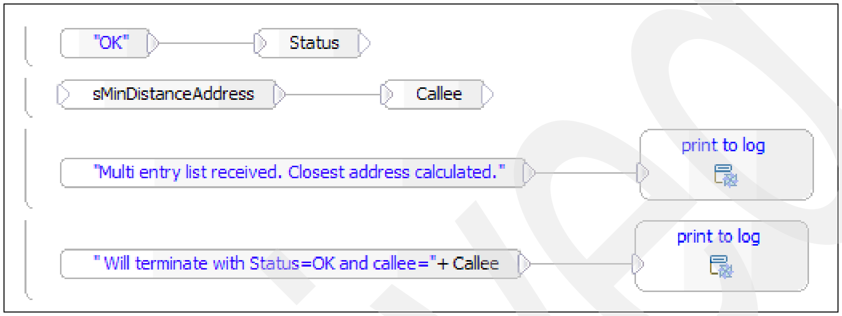


Figure 12-57 Assign Closest Address snippet

At this stage we have calculated the closest address and we are ready to return a response to the initial request.

Implement the Return Reply sequence

This sequence responds to the initial request by returning a status and the address that is closest to the originator. The Reply node was automatically created when we created the business process. We just have to add some tracing information.

1. Select the Reply node.
2. Right-click and select **Insert Before** → **Snippet**.
3. Rename the new snippet `Log Reply`.
4. With the snippet is selected, open the Properties view.
5. Select the Details tab.
6. Create a visual snippet as shown in Figure 12-58.

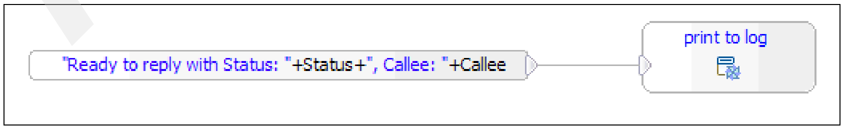


Figure 12-58 Log Reply snippet

Implement the Establish Call sequence

This sequence invokes the Third Party Call Control service to establish a call between the originator and the closest address

Before we invoke the Third Party Call Control service we need to define the required input and output variables. Using the Add icon next to the Variables label add the following variables:

Table 12-9 Variables for Establish Call sequence

Variable	Data type
MakeCallParameters	makeCall
MakeCallResponse	makeCallResponse
bMakeCallFault	boolean

Follow these steps to make a call.

1. Select the Establish Call sequence, and right-click.
2. Select **Add** → **Choice**.
3. Rename the new choice node `Establish Call Choice`.
4. Select the default case.
5. Open the Properties view and select the Description tab.
6. Enter `Status OK` as Display Name.
7. Select the Details tab.
8. Click **Create a New Condition**. This will open the visual editor.
9. Create a condition as shown in Figure 12-59.

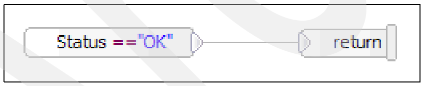


Figure 12-59 Establish Call condition

10. Select the Status OK case.
11. Right-click and select **Add** → **Snippet**.
12. Rename the new snippet `Assign MakeCall Input`.
13. With the snippet selected, open the Properties view .
14. Select the Details tab.
15. Create a visual snippet as shown in Figure 12-60.

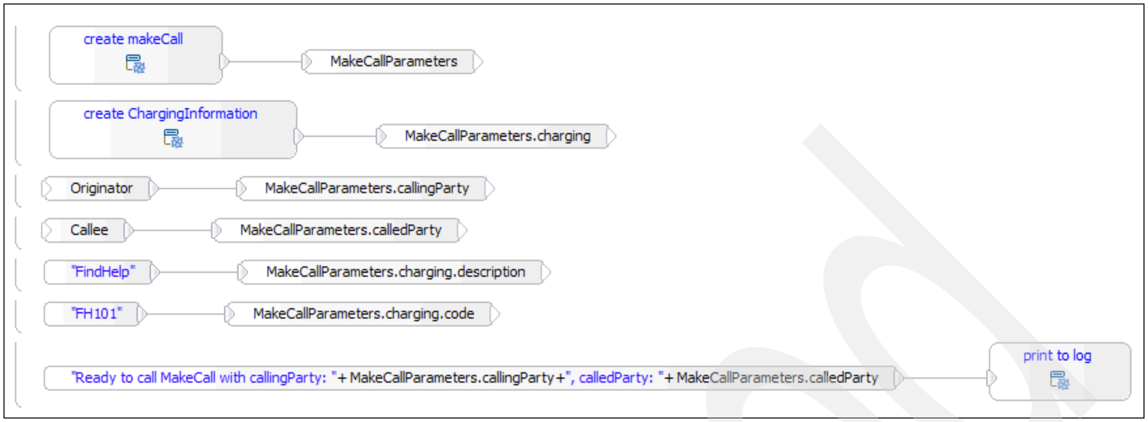


Figure 12-60 Assign MakeCall Input snippet

Add the Otherwise condition to cover the error case

16. Select the Establish Call choice node.
17. Right-click and select **Add Otherwise**.
18. Select the Otherwise case.
19. Right-click and select **Add** → **Snippet**.
20. Rename the new snippet `Log No Call`.
21. With the snippet selected, open the Properties view .
22. Select the Details tab.
23. Create a visual snippet as shown in Figure 12-61.

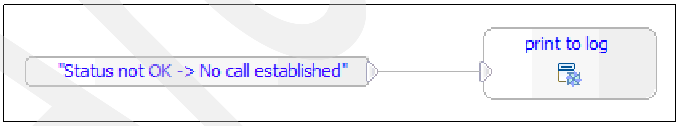


Figure 12-61 Log No Call snippet

We are ready to invoke the makeCall operation of the Third Party Call Control Web service.

24. Select the Status OK case.
25. Right-click and select **Add** → **Invoke**.
26. Rename the new Invoke node `MakeCall`.
27. With the Invoke node selected, open the Properties view.

28. Select the Details tab.

29. Click **Browse** to select a Reference Partner. The partner selection dialog box will be opened.

30. Select **ThirdPartyCallControlPartner**.

31. Click **OK**.

32. The selected Partner and Interface are displayed in the Details tab.

33. Choose **makeCall** as the operation.

34. Make sure that **Use Data Type Variables** check box is checked.

35. Similar to the getLocation Invoke node (see Step 24 on page 400) assign the following variables to the input and output parameters of the makeCall operation:

Table 12-10 Assign variables to the makeCall parameters

Parameter	Variable
parameters	MakeCallParameters
result	MakeCallResponse

When finished, the MakeCall Details tab should look like Figure 12-62 on page 409.

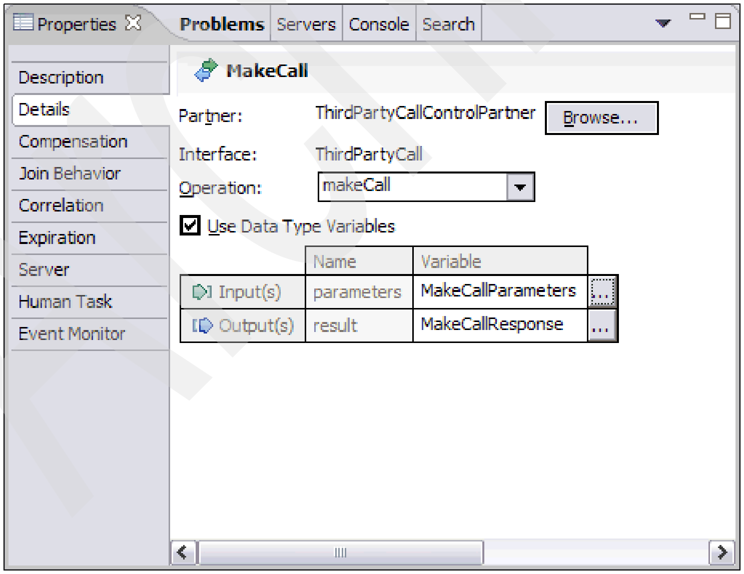


Figure 12-62 MakeCall details

The ThirdPartyCall interface defines special faults in the operation signature. We need to catch these faults. The next sequence of steps will add a fault handler to the Invoke activity:

36. Select the MakeCall activity.
37. Right-click and select **Add Fault Handler**. A new fault handler activity will be added to the right of the MakeCall activity.
38. With the new fault handler selected, open the Properties view.
39. Select the Details tab and apply the following settings:
    a. Fault Type: Choose `User-defined`.
    b. Namespace: Select: `http://www.csapi.org/wsdl/parlayx/third_party_call/v2_1/interface`
    c. Fault Name: Select `ServiceException`.
    d. Variable Name: Enter `MakeCallServiceException`.
    e. Choose `Data-Type`.
    f. Click **Browse**.
    g. Select **ServiceException** from the Data Type Selection dialog box.

Next we add the exception handler to the Catch clause.

40. Right-click the ServiceException catch clause.
41. Select **Add** → **Snippet**.
42. Rename the new snippet `Handle MakeCallServiceException`.
43. With the snippet selected, open the Properties view.
44. Select the Details tab.
45. Create a visual snippet as shown in Figure 12-63.

Figure 12-63 Handle MakeCallServiceException snippet

These sequence of steps will log the MakeCall invocation.

46. Select the Status OK case.
47. Right-click and select **Add** → **Snippet**.
48. Rename the new snippet `Assign MakeCall Output`.

49. With the snippet is selected, open the Properties view.

50. Select the Details tab.

51. Create a visual snippet as shown in Figure 12-64.

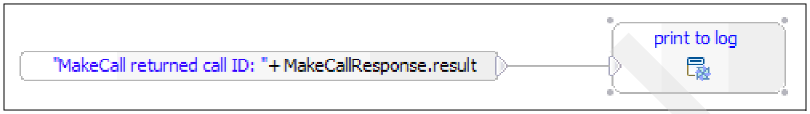


Figure 12-64 Assign MakeCall Output snippet

The Establish Call Choice node should now look like Figure 12-65 on page 411.

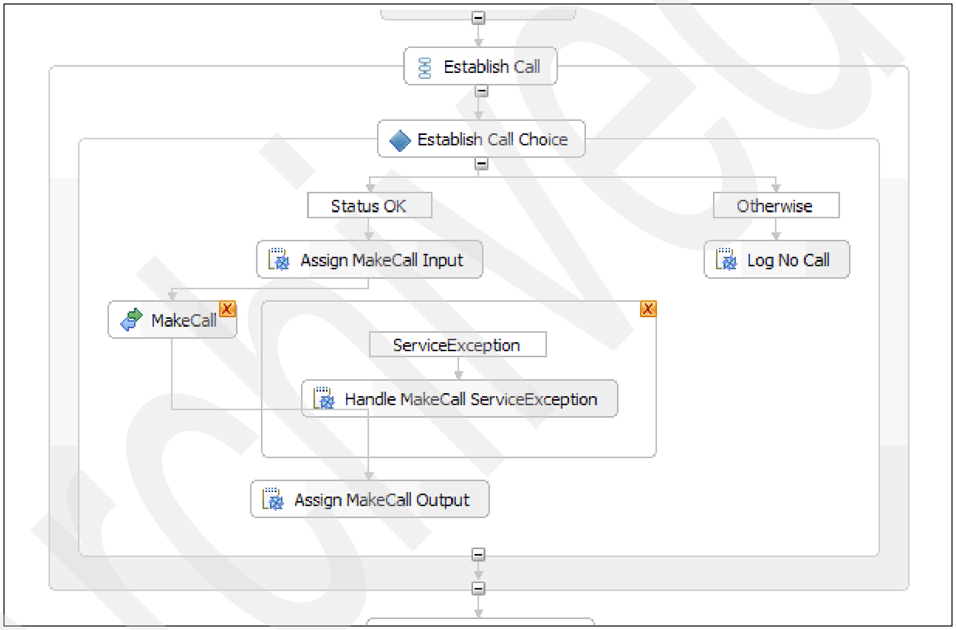


Figure 12-65 Establish Call sequence

Implement the Create Charging Event sequence

The next sequence of steps implement the invocation of the Diameter offline event charging service to generate charging events.

> **Note:** This is just a sample on how to integrate with a Diameter server. We will only supply the mandatory data to the charging service. Detailed discussion of the charging interface is outside the scope of this redbook.

Before we can invoke the Diameter event charging service we need to define the required input and output variables. Using the Add icon next to the Variables label add the following variables:

Table 12-11 Variables for Create Charging Event sequence

Variable	Data type
diaSessionId	string
diaRecordNumber	int
diaUserName	string
diaAccInterInterval	int
diaDestRealm	string
diaEventTimeStamp	long
diaOriginStateId	int
diaAct	Accounting
diaEventOffAccReturn	ACAResults

Follow these steps to create a charging event:

1. Select the Create Charging Event sequence.
2. Right-click and select **Add** → **Choice**.
3. Rename the new choice node `Create Charging Event Choice`.
4. Select the default case. Open the Properties view.
5. Select the Description tab.
6. Enter `Status OK` as Display Name.
7. Select the Details tab.
8. Click **Create a New Condition**. This will open the visual editor.
9. Create a condition as shown in Figure 12-66.

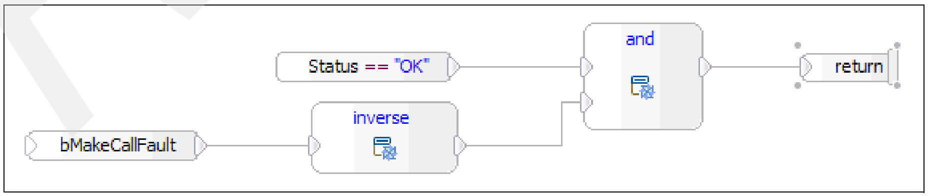


Figure 12-66 Create Charging Event condition

10.Select the Status OK case.

11.Right-click and select **Add** → **Snippet**.

12.Rename the new snippet `Assign ChargingEvent Input`.

13.With the snippet selected, open the Properties view.

14.Select the Details tab.

15.Create a visual snippet as shown in Figure 12-67 on page 413.

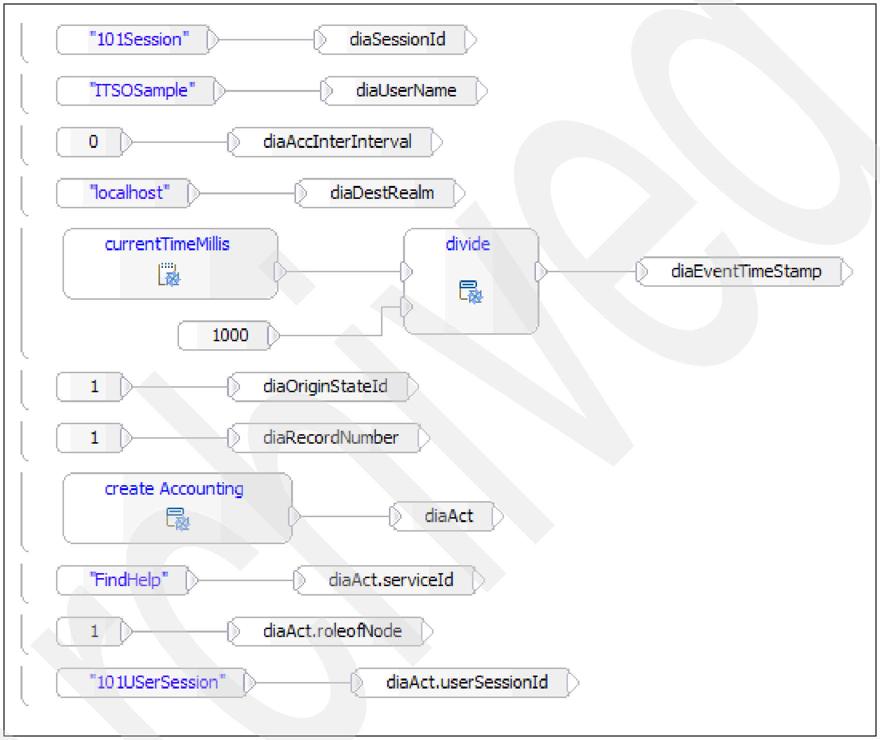


Figure 12-67 Assign ChargingEvent Input snippet

We add Otherwise condition to cover the error case.

16.Select the Create Charging Event Choice node.

17.Right-click and select **Add Otherwise**.

18.Select the Otherwise case.

19.Right-click and select **Add** → **Snippet**.

20.Rename the new snippet `Log No Charging`.

21.With the snippet selected, open the Properties view.

22. Select the Details tab.

23. Create a visual snippet as shown in Figure 12-68.

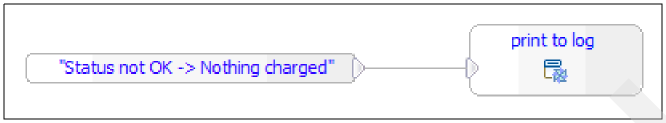


Figure 12-68 Log No Charging snippet

We are ready to implement the invocation of eventOfflineAccounting operation of the DiameterRfService Web service.

24. Select the Status OK case.

25. Right-click and select **Add** → **Invoke**.

26. Rename the new Invoke node `ChargeEvent`.

27. With the Invoke node selected, open the Properties view.

28. Select the Details tab.

29. Click **Browse** to select a Reference Partner. The partner selection dialog box will open.

30. Select **DiameterRfPartner**.

31. Click **OK**. The selected Partner and Interface will be displayed in the Details tab.

32. Choose **eventOfflineAccounting** as the operation.

33. Make sure that **Use Data Type Variables** check box is checked.

34. Similar to the getLocation invoke node (see Step 24 on page 400), assign the following variables to the input and output parameters of the eventOfflineAccounting operation:

Table 12-12 Assign variables to the eventOfflineAccounting parameters

Parameter	Variable
sessionId	diaSessionId
recordNumber	diaRecordNumber
userName	diaUserName
acctInterimInterval	diaAccInterInterval
destinationRealm	diaDestRealm
eventTimestamp	diaEventTimeStamp

Parameter	Variable
originStateID	diaOriginStateId
act	diaAct
eventOfflineAccountingReturn	diaEventOffAccReturn

When completed, the MakeCall Details tab should look like Figure 12-69 on page 415.

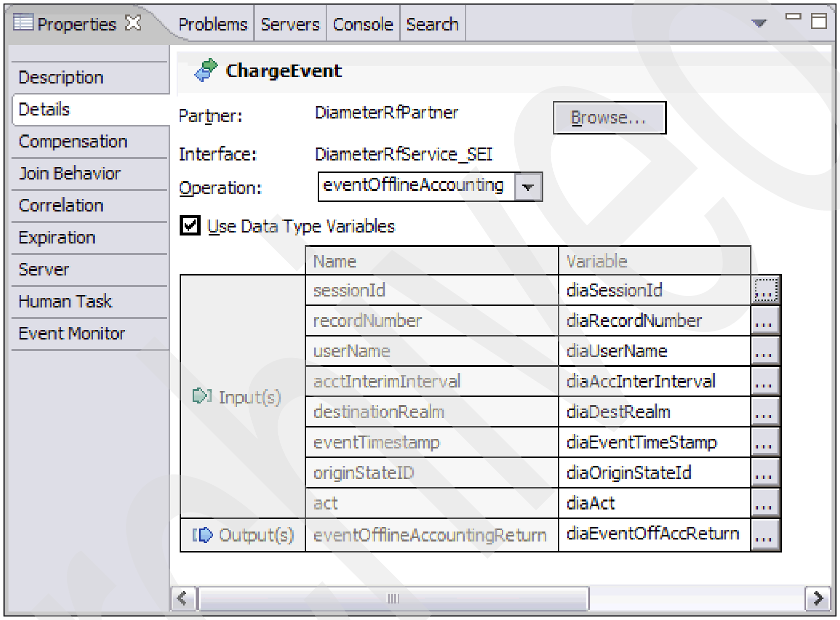


Figure 12-69 ChargeEvent details

35. Select the Status OK case.

36. Right-click and select **Add** → **Snippet**.

37. Rename the new snippet `Assign ChargingEvent Output`.

38. With the snippet selected, open the Properties view.

39. Select the Details tab.

40. Create a visual snippet as shown in Figure 12-70.

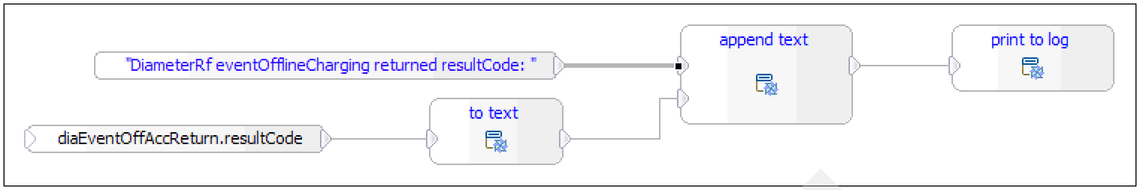


Figure 12-70 Assign ChargingEvent Output snippet

The Create Charging Event sequence should now look like Figure 12-71 on page 416.

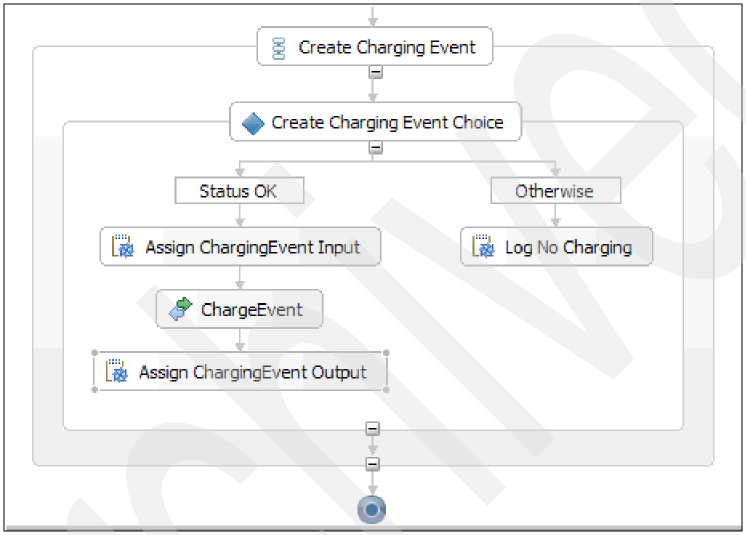


Figure 12-71 Create Charging Event sequence

12.3.8 Assemble the FindHelp module

After developing the components, we need to wire the components to complete the module.

> **Note:** The following steps are one of many possible approaches to building the assembly diagram.

Add components to the module

The assembly editor is the component of WebSphere Integration Developer where individual components are customized and wired to each other. To wire the components of FindHelp module together, perform the following steps.

1. Return to the Business Integration view.
2. Expand the FindHelp project.
3. Double-click the FindHelp module.
4. This opens the assembly editor in the workspace as in Figure 12-72 on page 417.

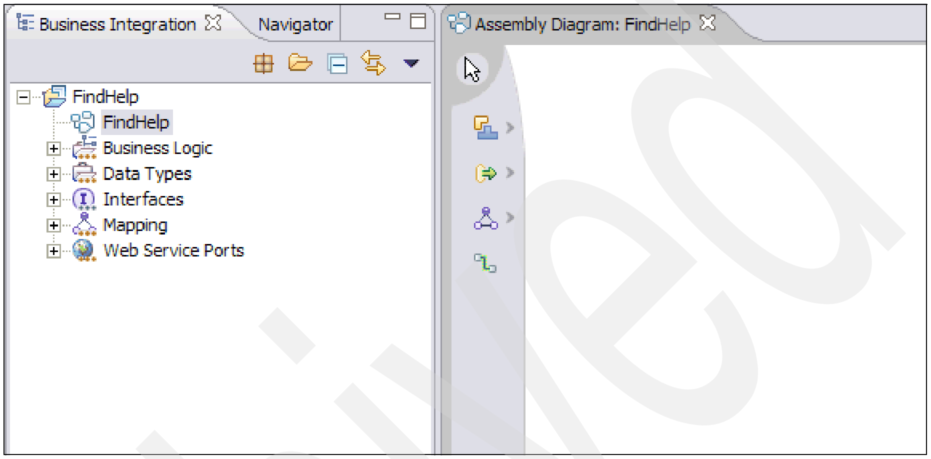


Figure 12-72 Assembly editor

> **Note:** The assembly editor consists of a canvas area and a palette for selecting components you can add to the diagram. The palette is located to the left of the canvas. There are nested items within the palette, you access these by clicking the gray > (greater than) symbol text to the parent item (Figure 12-72 on page 417).

5. Add a service component to the diagram by clicking the component icon .
6. Click anywhere on the canvas of the assembly diagram.
7. Select the Properties view.
8. Change the Display name and Name to `FindHelp`.

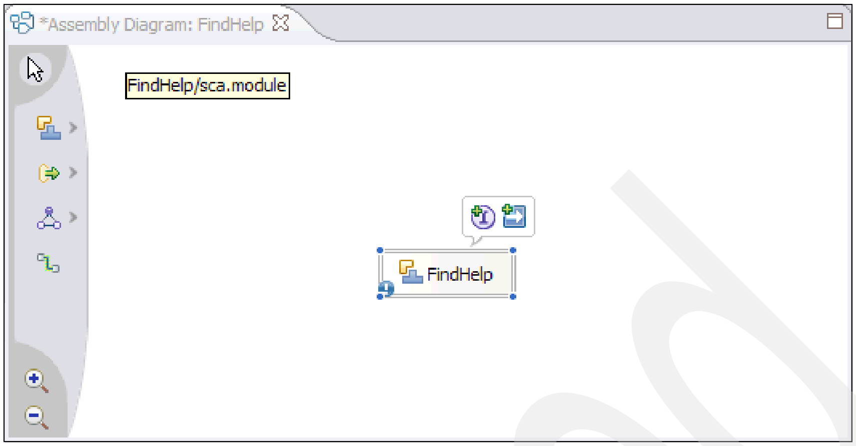


Figure 12-73 FindHelp component

9. In the assembly diagram, select the FindHelp component.
10. Click the Add Interface icon (see Figure 12-73).
11. When the Add Interface dialog opens, select **FindHelpInterface**.
12. Click **OK**.

We now need to define the implementation of the component. This is in fact the BPEL process we have created earlier in this section.

13. Right-click the **FindHelp** component.
14. Choose **Select Implementation** → **Process** as in Figure 12-74 on page 419.

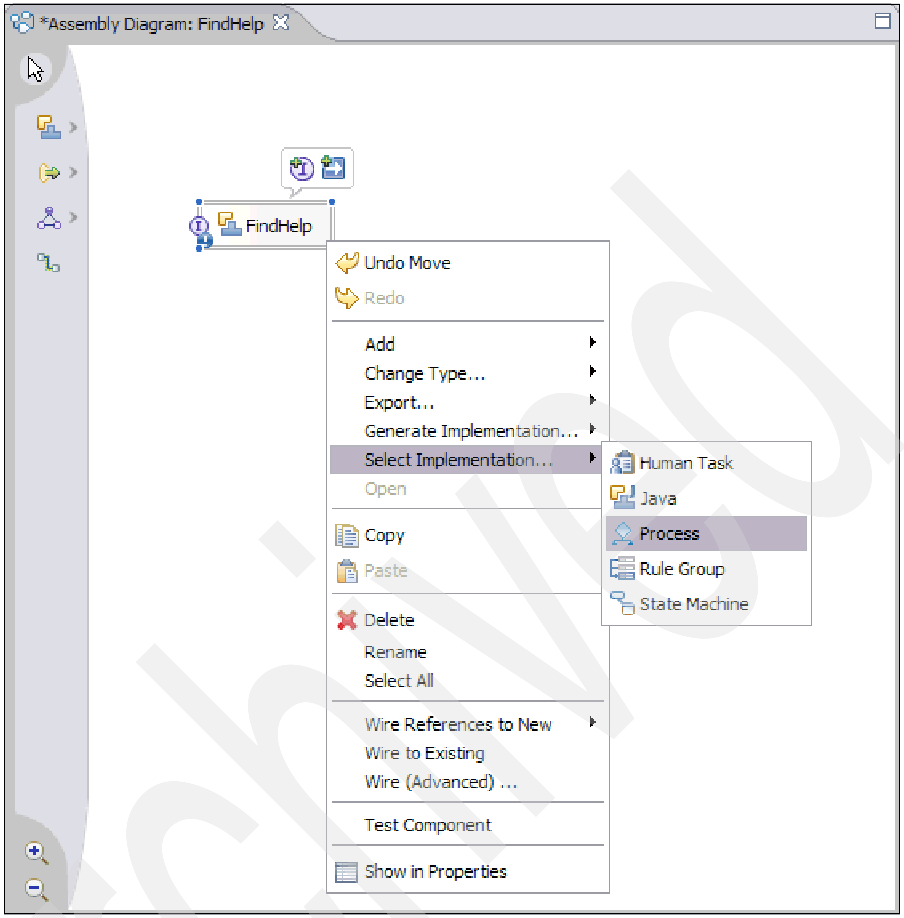


Figure 12-74 Select component implementation

15. In the process selection dialog select **FindHelpBP** process.
16. Click **OK**.

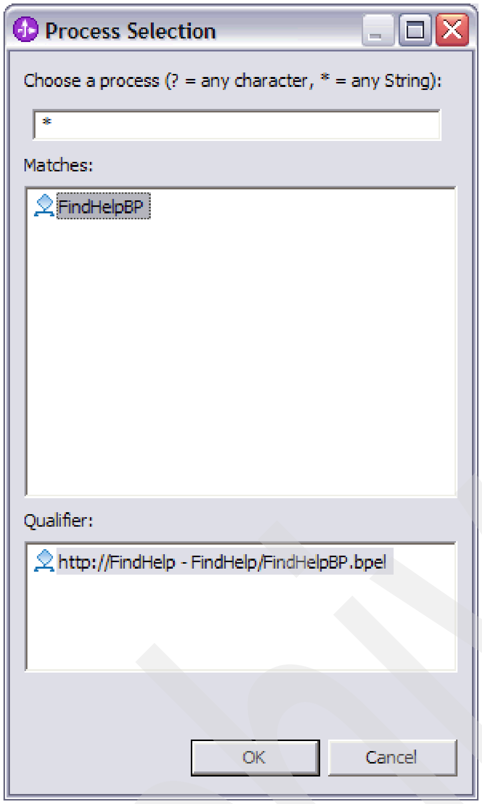


Figure 12-75 Select process

The FindHelp component is called by a SIP Servlet client. To allow this, you need to add an export reference to the component as follows:

17. Using the palette, select the export icon.

18. Add it to the canvas.

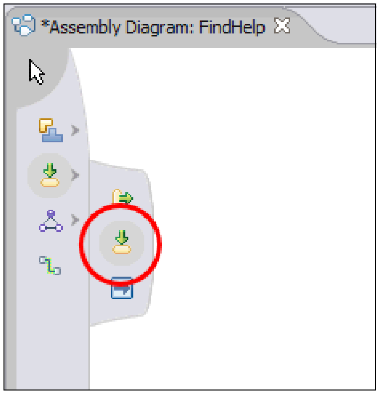


Figure 12-76 Add an export

19. Select the new Export and switch to the properties view.
20. Change the name to `setupcall`.
21. In the assembly diagram, select the setupcall export component.
22. Click the Add Interface icon .
23. In the Add Interface dialog select **FindHelpInterface**.
24. Click **OK**.

Now we need to wire both components.

25. In the assembly diagram, right-click the setupcall component.
26. Select **Generate Binding** → **Web Service Binding**.
27. Answer **Yes** to automatically generate a binding.
28. In the Select Transport dialog select **soap/http**.
29. Click **OK**.

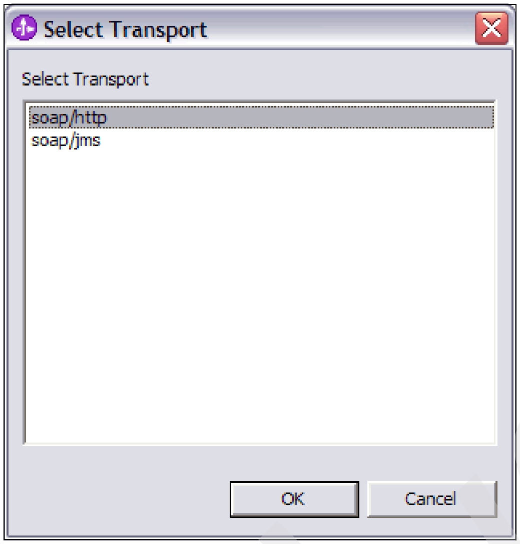


Figure 12-77 Select transport

30. In the assembly diagram, right-click the `setupcall` component.
31. Select **Wire to existing**. This will wire the setupcall component to the FindHelp component.

The assembly diagram should now look like the one in Figure 12-78 on page 423.

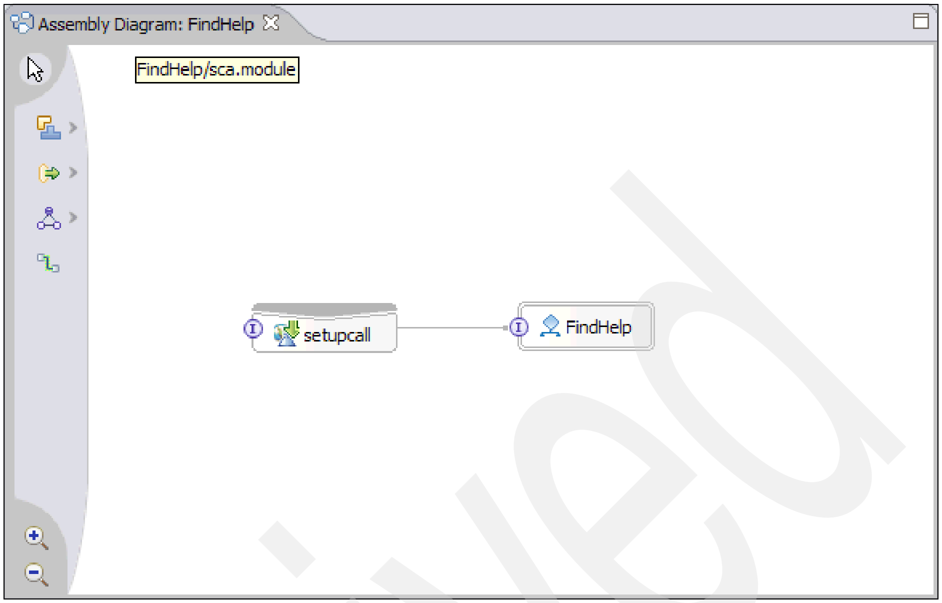


Figure 12-78 FindHelp wired to the setupcall export

Next we import and wire the external services that are required by the FindHelp component.

Add and wire the TerminalLocation

1. Using the palette, select the import icon and add it to the canvas.
2. In the Properties view, change the default name to `TerminalLocationImport`.
3. In the assembly diagram, select the TerminalLocationImport component.
4. Click the Add Interface icon .
5. In the Add Interface dialog, select **TerminalLocationImpl**.
6. Click **OK**.
7. Right-click the TerminalLocationImport component and select **Generate Binding → Web Service Binding**.
8. In the Binding tab of the Properties view click **Browse**. The Select WSDL file dialog opens.
9. Select **TerminalLocationImpl.wsdl**.
10. Click **OK**.

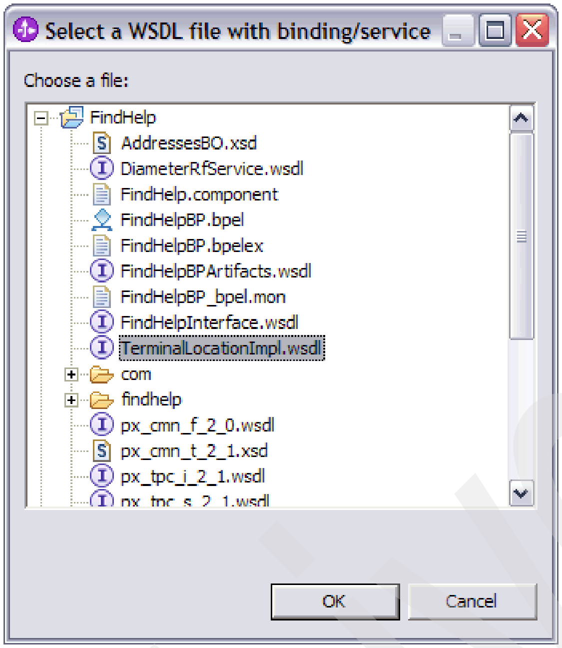


Figure 12-79 Select a WSDL file

The Binding tab in the Properties view should now look like Figure 12-80.

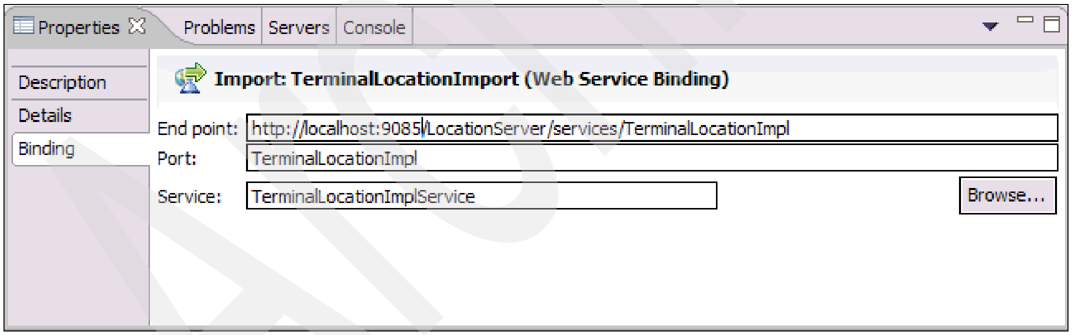


Figure 12-80 The binding properties of the import component

Note: Verify that the servername and port of the endpoint match your installation. If they don't, then change the endpoint accordingly.

We will now wire the FindHelp component to the TerminalLocationImport.

11.In the assembly editor, right-click the FindHelp component.

12. Select **Wire (Advanced)**.
13. In the Advanced Wiring dialog select **TerminalLocationImport**.
14. In the Add Wire dialog click **OK** to create a matching reference.
15. Click **OK**.
16. In the assembly editor, select the FindHelp component.
17. In the Details tab of the Properties view, expand References.
18. Click **TerminalLocationImplPartner**.
19. In the Details tab, change the partner reference Name to:
`LocationServicePartner`

> **Important:** The name of the reference of the component was generated. But it must match the partner reference we defined earlier in the business process editor (see 12.3.6, “Add partner references” on page 387).

20. In the menu select **Project** → **Clean**.
21. Accept the defaults.
22. Click **OK** to rebuild the project.

> **Attention:** If you get the error:
>
> ```
> “The operation 'invokeCallBack' of the interface
> 'ns1:FindHelpInterface' in the process component file
> '/FindHelp/FindHelp.component' does not specify the mandatory
> JoinTransaction interface qualifier.”
> ```
>
> Then, you need to specify the QoS Qualifier.
>
> a. In the assembly editor select the FindHelp component.
>
> b. In the Details tab of the Properties view, expand Interfaces.
>
> c. Select the **FindHelpInterface**.
>
> d. In the Qualifiers tab, select **Add**.
>
> e. In the Select Qualifier dialog select **Join transaction**.
>
> f. Click **OK**.

Add and wire the ThirdPartyCallControl

1. Using the palette, select the import icon  and add it to the canvas.
2. In the Properties view, change the default name to:
`ThirdPartyCallControlImport`

3. In the assembly diagram, select the ThirdPartyCallControlImport component.
4. Click the Add Interface icon .
5. In the Add Interface dialog, select **ThirdPartyCall**.
6. Click **OK**.
7. Right-click the ThirdPartyCallControlImport component and select **Generate Binding** → **Web Service Binding**.
8. In the Binding tab of the Properties view click **Browse**. The Select WSDL file dialog will open.
9. Select **px_tpc_s_2_1.wsdl**.
10. Click **OK**.
11. In the assembly editor, right-click the FindHelp component.
12. Select **Wire (Advanced)**.
13. In the Advanced Wiring dialog select **ThirdPartyCallControlImport**.
14. In the Add Wire dialog click **OK** to create a matching reference.
15. Click **OK**.
16. In the assembly editor, select the FindHelp component.
17. In the Details tab of the Properties view, expand References.
18. Click **ThirdPartyCallPartner**.
19. In the Details tab of the partner reference change the Name to: `ThirdPartyCallControlPartner`

Add and wire the Diameter Rf Service

1. Using the palette, select the import icon .
2. Add it to the canvas.
3. In the Properties view, change the default name to `DiameterRfImport`.
4. n the assembly diagram, select the DiameterRfImport component.
5. Click the Add Interface icon .
6. In the Add Interface dialog, select **DiameterRfService_SEI**.
7. Click **OK**.
8. Right-click the DiameterRfImport component.
9. Select **Generate Binding** → **Web Service Binding**.
10. In the Binding tab of the Properties view, click **Browse**.
11. Select WSDL file dialog opens.

12. Select **DiameterRfService.wsdl**.

13. Click **OK**.

14. In the assembly editor, right-click the FindHelp component.

15. Select **Wire (Advanced)**.

16. In the Advanced Wiring dialog select **DiameterRfControlImport**.

17. In the Add Wire dialog, click **OK** to create a matching reference.

18. Click **OK**.

19. In the assembly editor, select the FindHelp component.

20. In the Details tab of the Properties view, expand References.

21. Click **DiameterRfService_SEIPartner**.

22. In the Details tab of the partner reference change the Name to `DiameterRfPartner`.

The final assembly diagram should look like Figure 12-81.

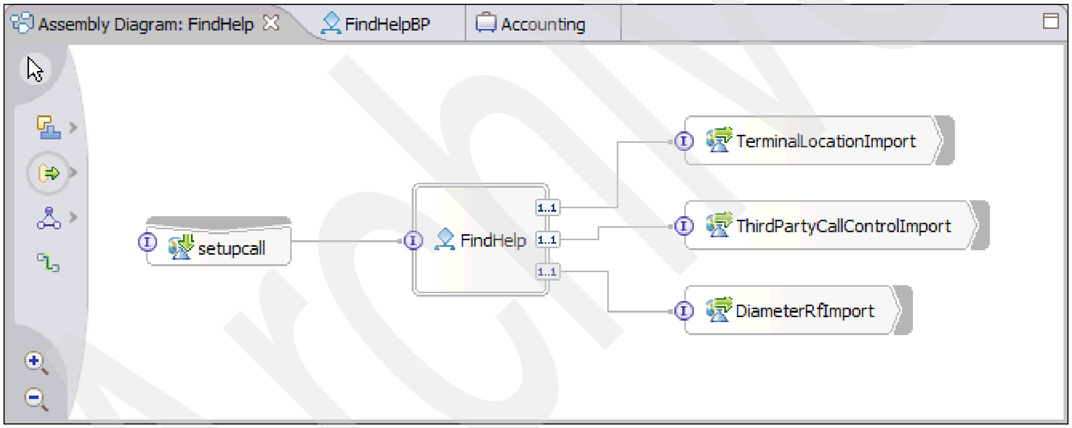


Figure 12-81 Final assembly diagram

12.4 Export the FindHelp WSDL files

We need to export the WSDL file, which describes the service that is provided by the FindHelp process component. This will be used to build the SIP Servlet for the FindHelp service. Refer to 12.2, “SIP Servlet development” on page 342 for how the FindHelp WSDL is used to implement the SIP Servlet.

Follow these steps to create the FindHelp WSDL archive.

1. From the menu select **File** → **Export**.

   This opens the Export wizard (seeFigure 12-82).
2. Select **ZIP file**.
3. Click **Next**.

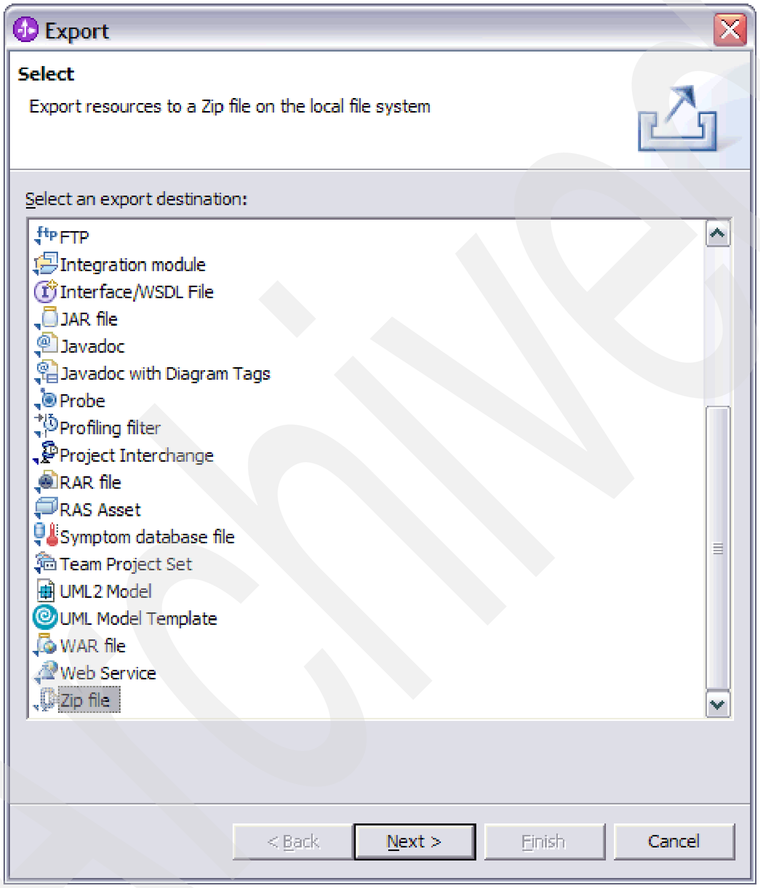


Figure 12-82 Export file wizard

4. In the next window, select the following files in the file list on the right:
   a. **AddressesBO.xsd**
   b. **FindHelpInterface.wsdl**
   c. **setupcall_FindHelpInterfaceHttp_Service.wsdl**

5. Specify the name of the archive and the directory where you want to store the archive in the “To zip file” field.
6. Click **Finish**.

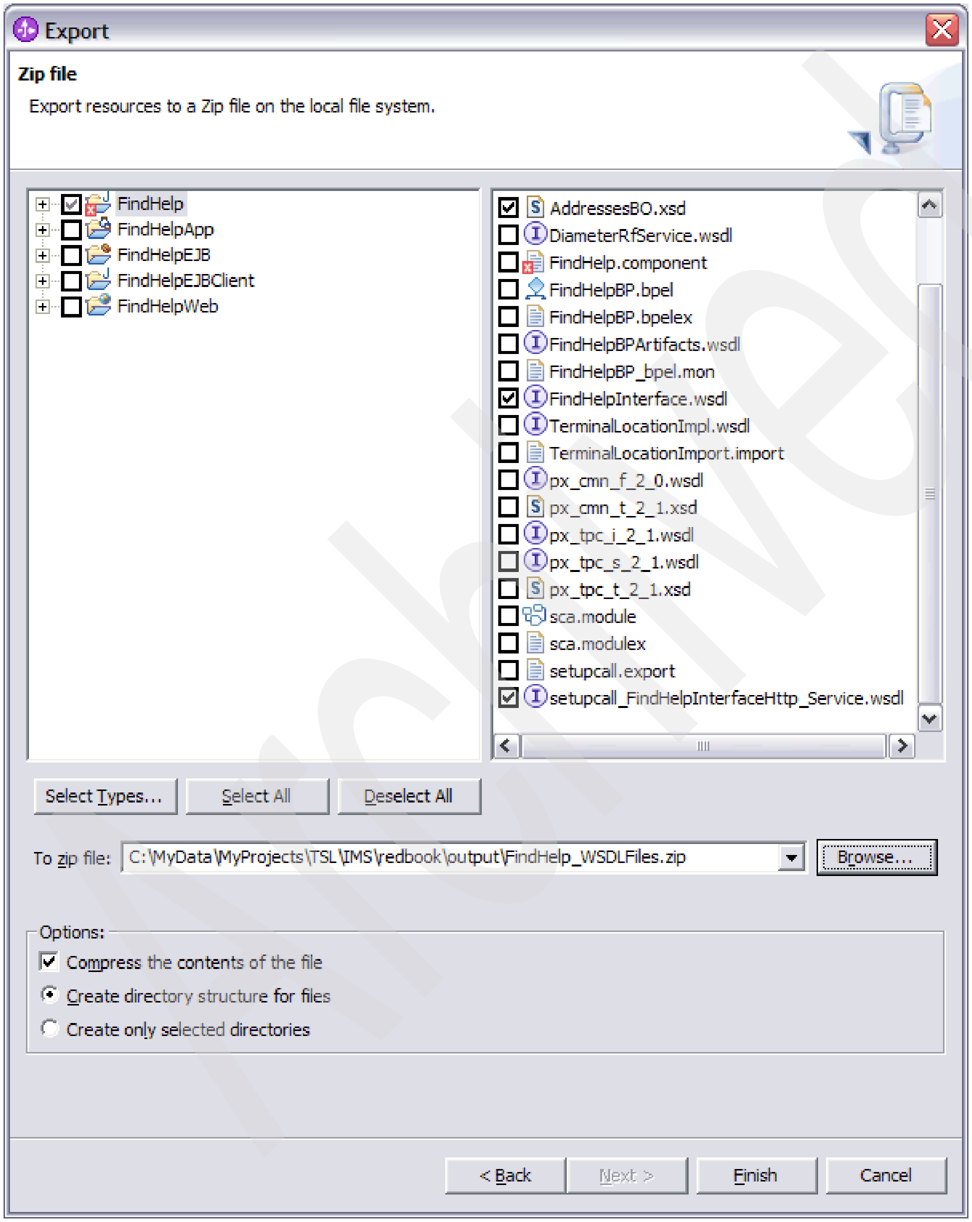


Figure 12-83 Select files for export

The FindHelp WSDL files are now available for the development of the FindHelp SIP Servlet.

12.4.1 Unit test the FindHelp module

The IBM WebSphere Integration Developer provides a comprehensive debugging and unit test environment. We make use of two of the debugging and unit testing components for the FindHelp business process unit test:

- The runtime/execution environment for the BPEL process provided by the WebSphere Process Server provides.
- The test scenarios provided by the Integrated Test Client.

Configuring the WebSphere Process Server

Before deploying any application to the process server you have to set the configuration parameters:

1. Select the WebSphere Process Server V6.0 runtime as your target test server as follows:
   a. Select **Windows** → **Preferences**.v
   b. Expand Server (in the left pane) and expand the list of installed runtimes.
   c. Select **WebSphere Process Server v6.0** as shown in Figure 12-84 on page 431.
   d. Click **OK**.

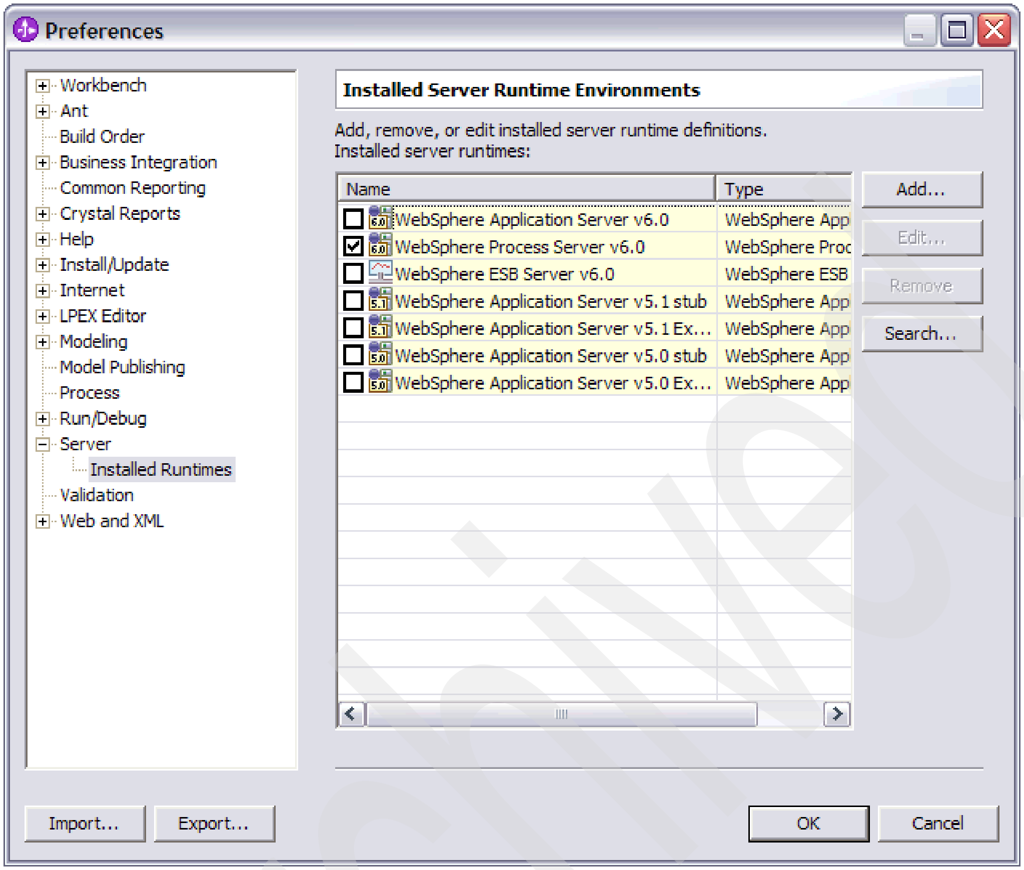


Figure 12-84 Installed runtimes

2. If it is not already open, open the Business Integration perspective.
3. Select **Window** → **Open Perspective** → **Business Integration**.
4. Select the Servers tab in the pane at the bottom right.

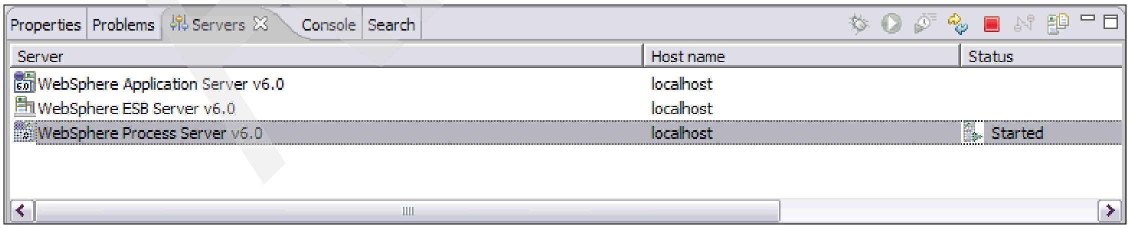


Figure 12-85 Servers view

5. Double-click the server name to open the configuration editor.
6. In the Server setting, select **SOAP** as the Server connection type and admin port.
7. In the Publishing section, select **Run server with resources within the workspace**.

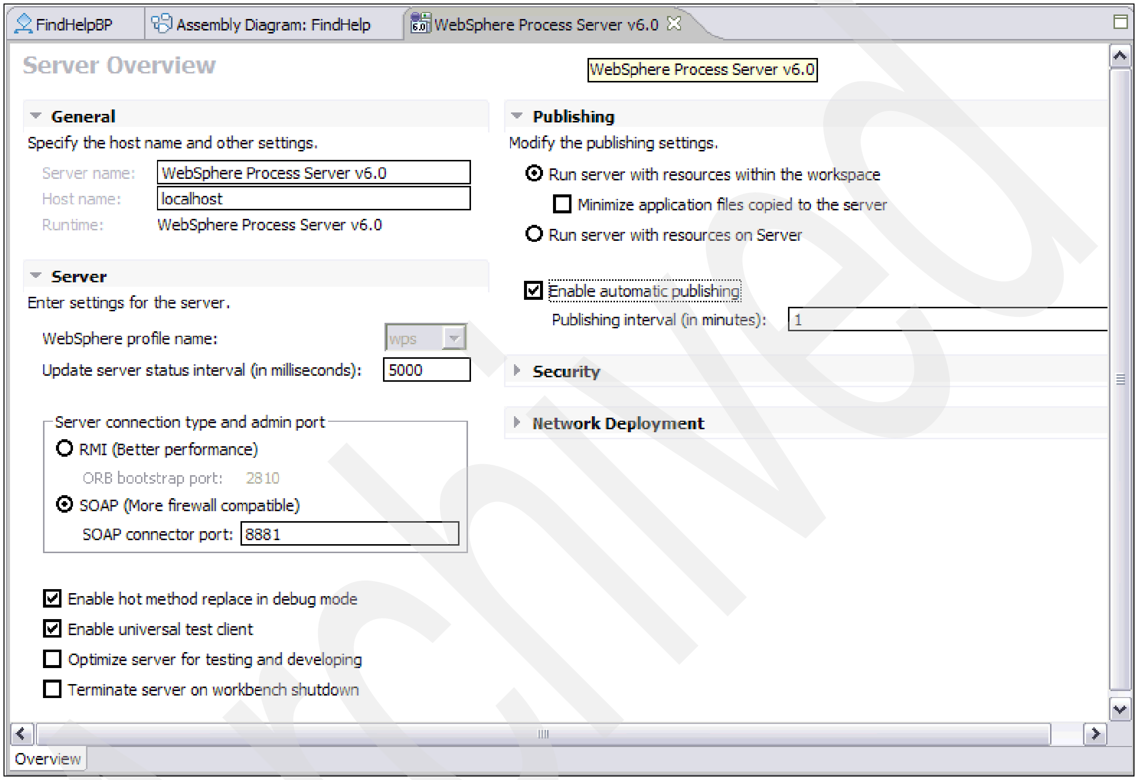


Figure 12-86   Using the server configuration editor

8. Save these settings and close the configuration editor.

Test the integration service module

The end-to-end Test Framework can be invoked on any service component from the assembly diagram.

Start by opening the assembly diagram:

1. From the Business Integration perspective, expand the FindHelp project.
2. Double-click the FindHelp module to open the assembly diagram.

3. Right-click the FindHelp component and select **Test Component** from the context menu.
4. The test editor similar to Figure 12-87 will open.

> **Note:** To run the interface test, select the import component whose interfaces you want to test. Right-click and select **Test Component**.
>
> What you are going to invoke is on the left column. On the right, you can select the module, component, interface, and operation that you want to invoke. Below that, are the parameters. You are able to provide values for the operation input message. Note also the Datapool option, it enables you to save and reuse input data if you must test a component several times and to manage sets of test data.

5. Provide values for the input parameters.
6. Click **Continue**.

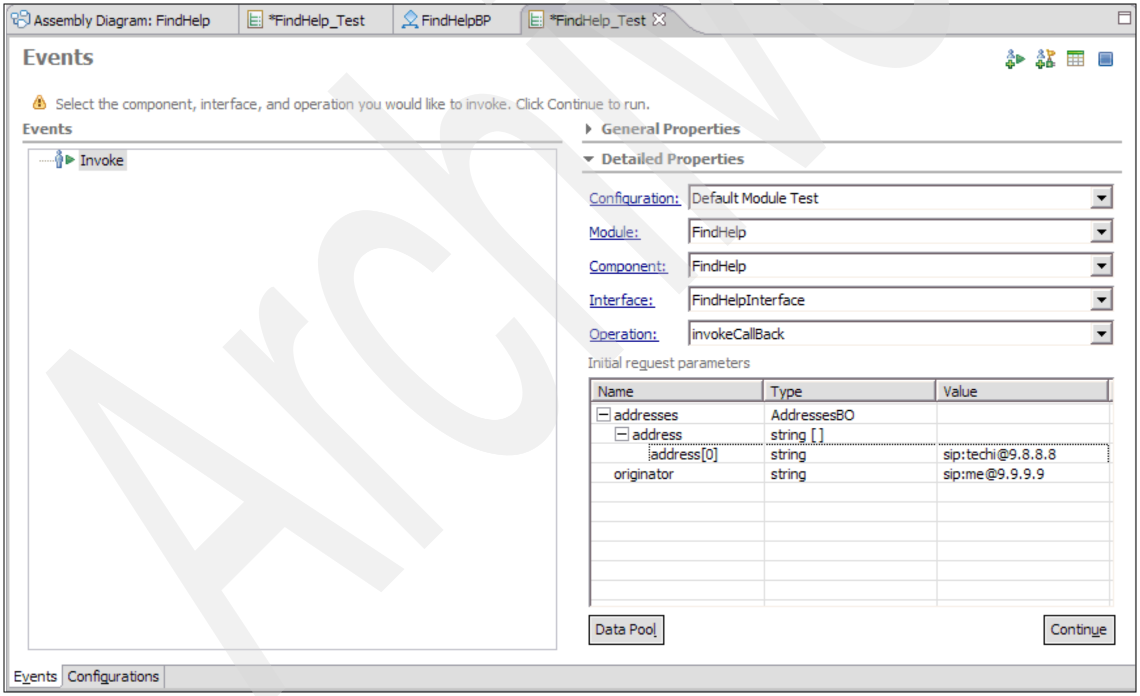


Figure 12-87 Test editor

7. In the Deployment Location dialog, verify that **WebSphere Process Server v6.0** is selected.

8. Click **Finish**.

Once the test has completed, you can see the response message in the Detailed Properties pane (Figure 12-88).

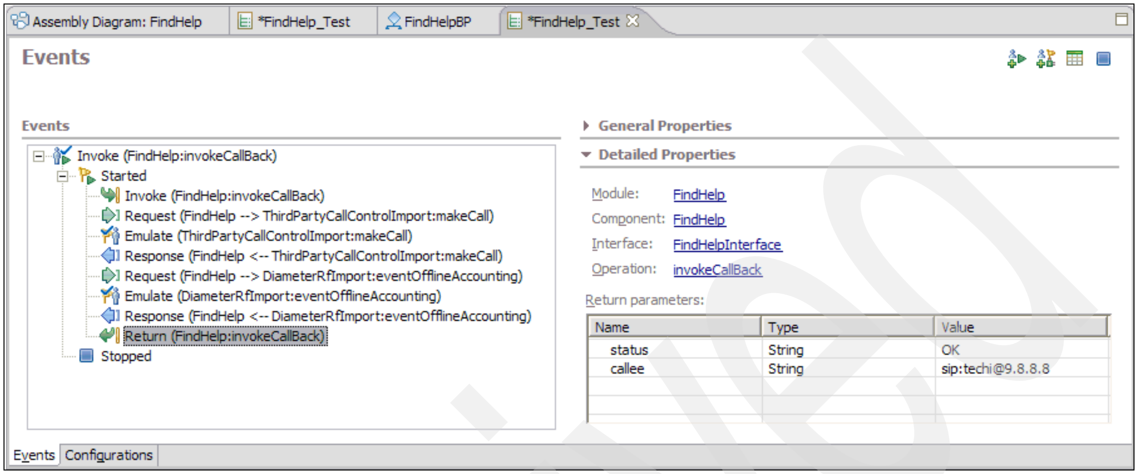


Figure 12-88 Test results

In the events pane, you can see the successful invocations and responses. The Configurations pane displays the details of the module that you have been testing (see Figure 12-89 on page 435). Notice the concept of an emulator. If the module diagram contains interfaces for the implementation that are not yet ready for testing, you can ask to emulate that component. This gives the you the option to provide output using the test facility, instead of invoking the actual implementation of that interface.

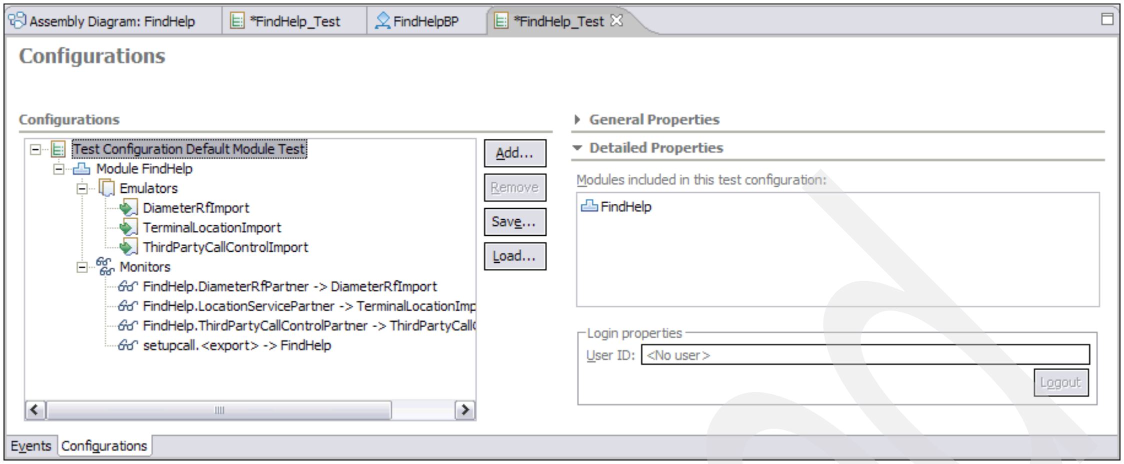


Figure 12-89 Test configuration editor

12.5 The location simulator

A location server is required for the sample application to retrieve the location-based information for the originator of the service and available members of the group. This information is compared against the location of the originator to determine the closest resource. As many readers will not have access to a real location server a location simulator has been developed that exposes a Web service interface. We do not discuss the design and development of the location simulator in this redbook, however the source code is included in the additional material section for installation and review. In this section, will provide a high level description of the implementation to assist you with the replication of the demo scenario.

The location simulator includes the following methods that can be invoked either by a Java client or Web Service consumer such as a BPEL process (Figure 12-90 on page 436 shows the class diagram):

- getLocation

  Provides the ability to specify an URI with a requested and accepted accuracy. If the acceptable accuracy cannot be met or the URI is not found in the database, a service exception will be returned. Successful processing of the request returns a LocationInfo object which includes variables such as the latitude and longitude. This method is used by the BPEL to retrieve the longitude and latitude of all the available members of the group.

- getTerminalDistance

  Provides the ability to specify a URI, latitude and longitude. The server will then lookup the current location for the specified URI and calculate the distance between the current location and the specified longitude and latitude. The method is called with the data returned from the getLocation method to calculate the distance between the originator and available members of the group. The return value is an integer that represents the number of meters between the URIs current location and the specified longitude and latitude.

- getLocationForGroup

  Is a convenience function that allows multiple URIs to be specified for a given requested and acceptable accuracy. Functionally there is no difference between calling the getLocation method multiple times and calling getLocationForGroup once.

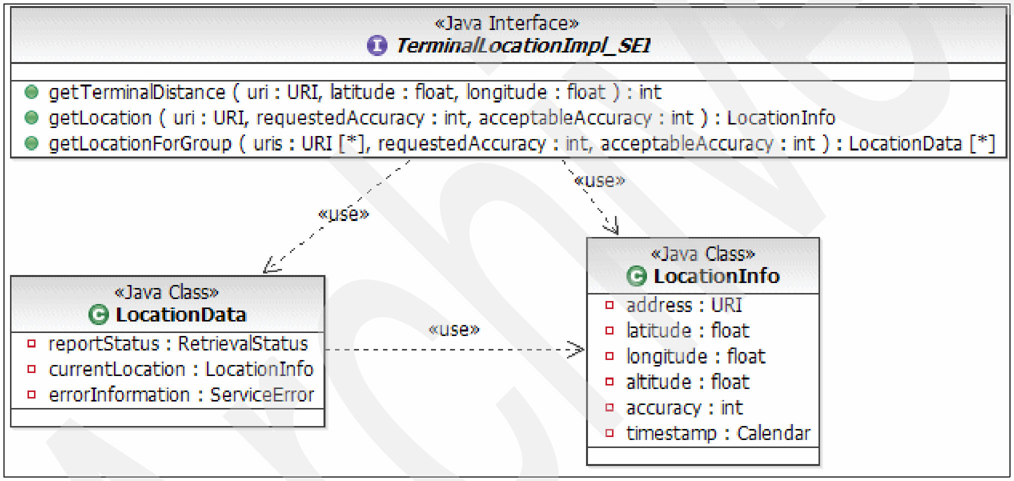


Figure 12-90 Location server class diagram

The data store behind the location simulator is a flat file that details the location information for URIs. This file is called *terminallocation.db* and stored in *c:* for Windows and */root* for Linux. An example configuration file to customize can be located from Appendix C, “Additional material” on page 637, and an example is shown in Example 12-10.

Example 12-10 Example terminallocation.db file

```
# this file contains the terminal locations, fields are # separated
# lines beginning with a # are treated as comments
# the field order is URI, latitude, longitude, altitude, accuracy
# this file will need to be customized to meet the installation
```

```
# requirements
sip:caller@9.42.170.160:5070#89.89#12.12#1278#1
sip:philippe@9.42.170.160:5998#89.89#12.12#1278#1
sip:callum@9.42.170.160:5999#89.89#62.12#1278#1
sip:cameron@9.42.170.160:5060#89.89#12.12#1278#1
sip:sipphone@9.42.171.130#89.00#10.10#1278#1
sip:sipp@9.42.171.130#89.00#10.19#1278#1
sip:callum@9.42.171.135#89.89#12.13#1278#1
sip:rebecca@9.42.171.135#89.89#12.15#1278#1
```

Information regarding the installation and configuration of the location server can be found in 13.2.2, “Location server setup” on page 448.

13

Sample IMS application test environment

This chapter provides an overview of the test environment that was set up and used to test the FindHelp sample IMS application. It describes the necessary configuration for setting up the environment, running test scenarios and for problem determination and resolution.

This chapter contains the following:

- Overview of the test environment
- Setting up the test environment
- Executing the test scenarios
- Problem determination and resolution

13.1 Overview of the test environment

The test environment allow for end-to-end testing of the FindHelp sample IMS application. Our test scenario include the execution of the following use cases:

- Administrator adds service topic
- Technician's publishing of presence information
- Caller request for FindHelp service topic LockOut

To execute these use cases, we created a test environment that included both Linux and Windows systems. The Linux test server hosted various IMS components and the FindHelp SIP Servlet. Installed on the Windows test machine were the simulated SIP traffic clients, SIP telephony clients and the BPEL engine containing the FindHelp BPEL process.

The following products and components were installed on the Linux test machine:

- IBM WebSphere Application Server Version 6.1

  Provides a converged HTTP/SIP application server for the SIP and IMS components.

- IBM WebSphere Presence Server Version 6.1.0

  Group list management and presence information is provided by the IBM WebSphere Presence Server which includes the IBM WebSphere Group List Server.

- IBM WebSphere Telecom Web Services Server Version 6.1.0

  For the testing of the IMS sample application we use only the Parlay X Third Party Call Control functionality of the IBM WebSphere Telecom Web Services Server for the call setup between the SIP softphones.

- IBM WebSphere IP Multimedia Subsystem Connector Version 6.1.0

  Provides the ISC Diameter Rf interface for simulated offline billing.

The WebSphere Process Server was installed on the Windows test machine. It provided a BPEL execution environment for the FindHelp business process logic.

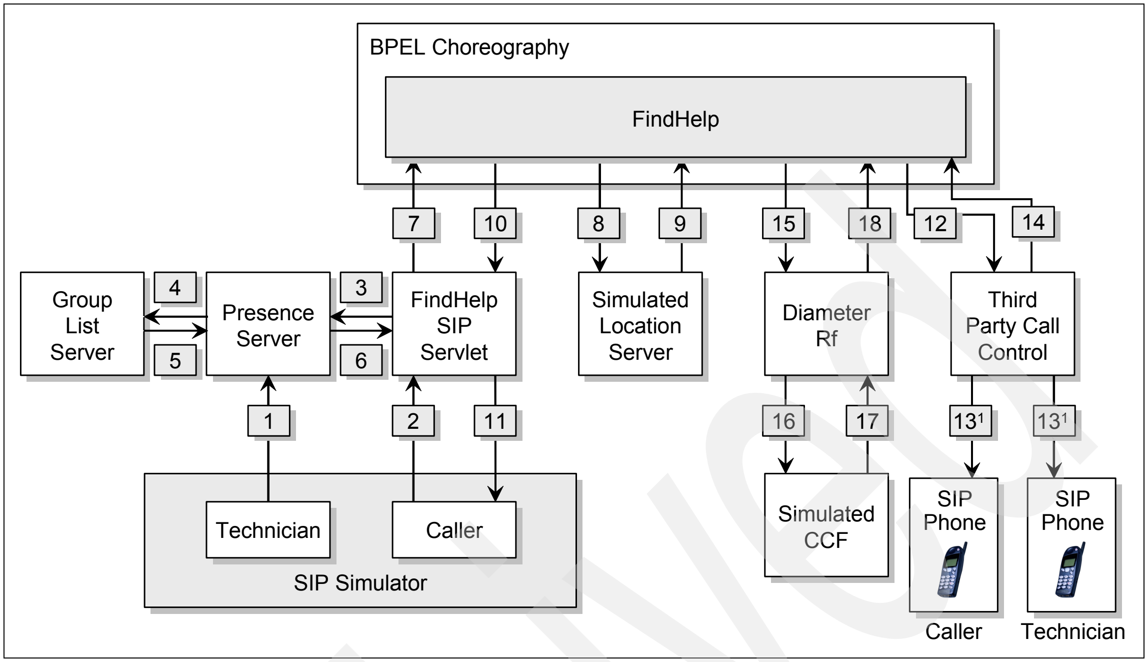


Figure 13-1 Test Environment Component Overview

A number of simulated components were also installed in the test environment:

- IMS components

  Two executable jar files for simulating IMS components were installed in the test environment:

  – Simulated Location server
  – Simulated CCF

- SIP traffic

  SIPp from Source Forge and SIPp XML scripts were used to generate the SIP traffic for simulating clients.

- SIP phones

  For the call-setup and voice traffic simulation we used two SIP softphones, SJphone and sipXphone. Two different phones are necessary as they were installed on the same Windows test machine. Two instances of the same SIP softphone could not be started on the same machine.

Note: Some of the components can be downloaded as part of the additional materials for this redbook (see Appendix C, “Additional material” on page 637). It includes the following:

- SIPp scripts
  - Simulated Technician SIPp scripts
  - Simulated Caller SIPp scripts
- FindHelp components
  - BPEL Flows
  - SIP servlet

The hardware configuration for the test environment consisted of a Windows client and a Linux server setup. Figure 13-2 on page 444 provides an overview of the components deployment on the Windows and the Linux test machines.

- Windows client

  The hardware used for the Windows test machine was an IBM Thinkpad T42 with a 1.8 GHz Intel Pentium Processor and 2GB RAM running Windows XP Professional with Service Pack 2 applied. The available disk space was 20 GB.

  Note: The minimum requirement for the client hardware is listed here:

  - 15 GB free disk space
  - 2 GB RAM
  - Pentium 1.6 GHz

- Linux server

  The hardware configuration for the Linux test server consisted of the IBM eServer xSeries® 345 Server with 3GHz Intel Xeon® Processor and 4 GB RAM running Red Hat Enterprise Linux AS 4.0 Update 3. The available disk space was 200 GB.

Note: Hardware requirements

The following information represents the minimum requirements. For greater performance and scalability, additional hardware may be needed.

- Linux on pSeries®

  IBM WebSphere IP Multimedia Subsystem Connector V6.1, IBM WebSphere Presence Server V6.1, and IBM WebSphere Telecom Web Services Server V6.1 only supports NEBS-compliant IBM eServer pSeries servers.

  – Processor: Power 5, 1.2 GHz (32- and 64-bit)
  – Physical memory: 2 GB recommended
  – Disk space: 20 GB of free space v
  – Other: CD-ROM or access to shared network drive where CD images are available

- Linux on Intel

  IBM WebSphere IP Multimedia Subsystem Connector V6.1, IBM WebSphere Presence Server V6.1, and IBM WebSphere Telecom Web Services Server V6.1 supports BladeCenter® for Intel x86 platforms

  – Processor: Pentium 4, 2.4GHz (32- and 64-bit)
  – Physical memory: 2 GB recommended
  – Disk space: 20 GB of free space
  – Other: CD-ROM or access to shared network drive where CD images are available

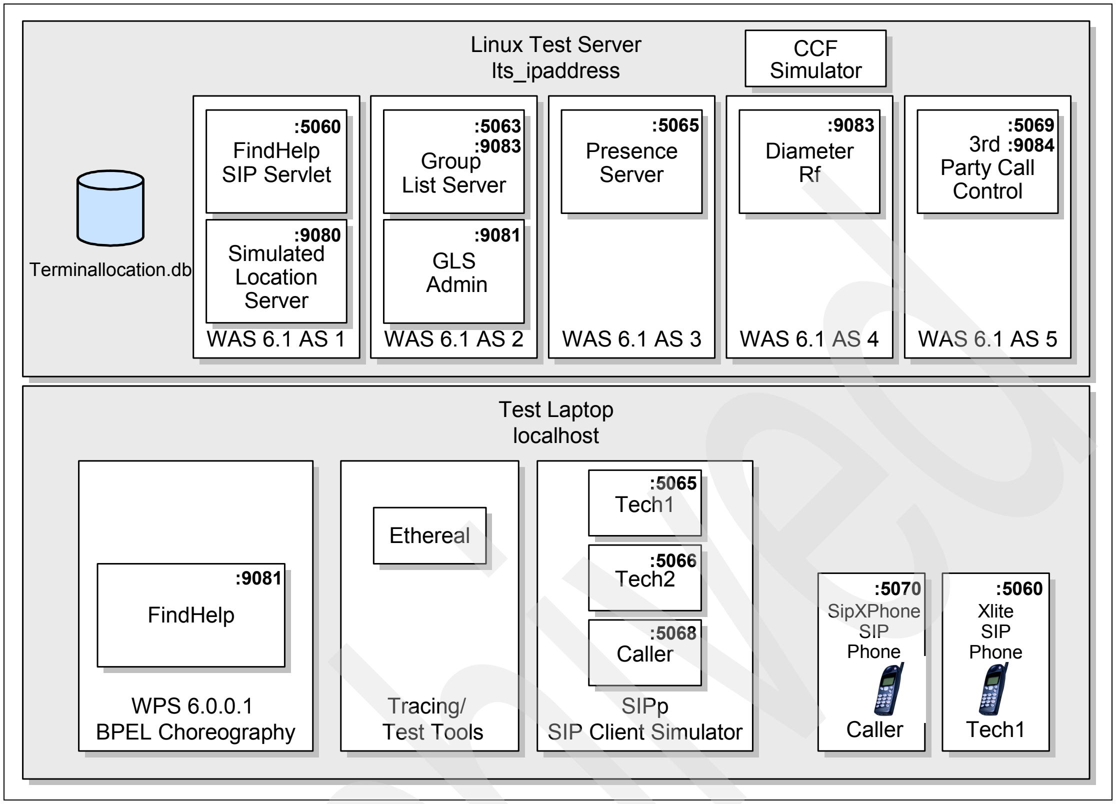


Figure 13-2 Overview of installed components

Table 13-1 provides a listing of the ports used by the components in the test environment. You may use other ports, however the setup and configuration instructions for the components assume that these ports are used. If you use alternative ports, you must replace the ports in the instructions with the actual ports in your configuration.

Table 13-1 Test environment port usage

Component	**IP-Address**	**SIP Traffic Port**	**HTTP Traffic Port**	**Diameter Traffic Port**
FindHelp SIP Servlet	<lts_ipaddress>	:5060		
Simulated Location	<lts_ipaddress>		:9080	
Group List Server	<lts_ipaddress>	:5063	:9081 (xcap)	
GLS Admin Console	<lts_ipaddress>		:9081	

Component	IP-Address	SIP Traffic Port	HTTP Traffic Port	Diameter Traffic Port
Presence Server	<lts_ipaddress>	5065		
Diameter Rf	<lts_ipaddress>		:9083	3868
CCF Simulator	<lts_ipaddress>			3868
3rd Party Call Control	<lts_ipaddress>	:5069	:9084	
FindMe BPEL	<wtl_ipaddress>		:9081	
Tech1 SIPp	<wtl_ipaddress>	:5065		
Tech2 SIPp	<wtl_ipaddress>	:5066		
Caller SIPp	<wtl_ipaddress>	:5068		
Tech1 SIP softphone	<wtl_ipaddress>	:5060		
Caller SIP softphone	<wtl_ipaddress>	:5070		

13.2 Setting up the test environment

The step-by-step instructions for installing the different components in the test environment is described in Appendix B, "Installing the sample application test environment" on page 531. In this section we describe the necessary configuration that you need to perform to set up the test environment.

13.2.1 Group List Server setup

The Group List Server is configured using resource_list and rls_services documents that describe the group and group members to be used by the sample application.

The file lockout_techies.xml contains the list of members that form the group lockout_techies. This file is available as part of the additional materials that you can download for this redbook (see Appendix C, "Additional material" on page 637). Example 13-1 on page 446 shows the listing of the lockout_techies.xml document.

The <list> element contains 5 <entry> elements, each of these identifying a technician and his SIP URI.

Example 13-1 Lockout resource_list document

```
<?xml version="1.0" encoding="UTF-8"?>
<rl:resource-lists xmlns:rl="urn:ietf:params:xml:ns:resource-lists">
<rl:list name="lockout_techies">
   <rl:entry uri="sip:tech1@itso.ral.ibm.com">
       <rl:display-name>Cameron Martin</rl:display-name>
   </rl:entry>
   <rl:entry uri="sip:tech2@itso.ral.ibm.com">
       <rl:display-name>Callum Jackson</rl:display-name>
    </rl:entry>
    <rl:entry uri="sip:tech3@itso.ral.ibm.com">
       <rl:display-name>Philippe Bazot</rl:display-name>
    </rl:entry>
    <rl:entry uri="sip:tech4@itso.ral.ibm.com">
       <rl:display-name>Rebecca Huber</rl:display-name>
   </rl:entry>
   <rl:entry uri="sip:tech5@itso.ral.ibm.com">
       <rl:display-name>Jochen Kappel</rl:display-name>
   </rl:entry>
 </rl:list></rl:resource-lists>
```

You make use of the command line interface to create the members in the GLS server. The command is run in the directory where the GLS client was installed using the following syntax:

```
./xcap_put.sh -user username -password pwd -filename
<root_directory>/lockout_techies.xml -content_type
application/resource-lists+xml
http://hostname:port/services/resource-lists/users/username/lockout_
techies.xml
```

Where:

- username

  Is the name of a GLS user having the `admin` authority (in our case it is `GLSUser1`)

- pwd

  Is the chosen password for `username`

- <root_directory>

  Is the directory where `lockout_techies.xml` file has been stored

- hostname

  Is the server host or IP address of the GLS server

- port

  Is the port number to use for XCAP protocol (in this case it is 9081)

The rls_services document is named lockout_rls.xml. Its main purpose is to define the URI associated with the group of technicians which have skills to help customers with lockout problems. This file is also available as part of the additional materials that you can download for this redbook (see Appendix C, “Additional material” on page 637).

Example 13-2 shows the listing of the lockout_rls.xml document. The root element of an rls-services document is <rls-services>. It contains a <service> element with a single mandatory attribute, "uri". The URI (in this case it is sip:lockout@itso.ral.ibm.com) defines the resource associated with the group. It also contains a <resource-list> element which is the HTTP URI representing the XCAP element resource. Finally its contains a <packages> element which contains the presence SIP event package.

You must edit the file lockout_rls.xml document and substitute the value <lts_ipaddress> with the IP address of your Linux test server. If you use the same port assignments as in Table 13-1 on page 444, then the port number for the Group List Management Server will be left as 9081.

Example 13-2 rls_services document for FindHelp service topic: Lockout

```
<?xml version="1.0" encoding="UTF-8"?>
   <rls-services xmlns="urn:ietf:params:xml:ns:rls-services"
   xmlns:rl="urn:ietf:params:xml:ns:resource-lists"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <service uri="sip:lockout@itso.ral.ibm.com">

<resource-list>http://<lts_ipaddress>:9081/services/resource-lists/user
s/group1/lockout_techies.xml</resource-list>
     <packages>
     <package>presence</package>
     </packages>
   </service>
  </rls-services>
```

You also make use of the command line interface to create the document in the GLS server. It is run from the directory where the GLS client was installed using the following syntax:

```
./xcap_put.sh -user username -password pwd -filename
<root_directory>/redbook_rls.xml -content_type
application/rls-services+xml
```

```
http://hostname:port/services/rls-services/users/username/lockout_rls.xml
```

Where:

- username

  Is the name of a GLS user having the admin authority (in our case it is GLSUser1)
- pwd

  Is the chosen password for username
- <root_directory>

  Is the directory where redbook.xml file has been stored
- hostname

  Is the server host or IP address of the GLS server
- port

  Is the port number to use for XCAP protocol (in this case it is 9081)

13.2.2 Location server setup

You deploy the simulated Location server by importing the jar file that is included in the additional materials for this redbook (see Appendix C, “Additional material” on page 637). The following steps explain the instructions for deploying the simulated Location server code onto the Linux test server.

1. Open the Integration Solution Console for the WebSphere Application Server Node 1.
2. Click **Applications**.
3. Click **Install New Applications**.
4. Select the radio button **Remote File System**.
5. Click **Browse**.
6. Navigate to the file: `/root/opt/RBbespoke/`
7. Select the radio button for: `locationServerEAR.ear`
8. Click **OK**.

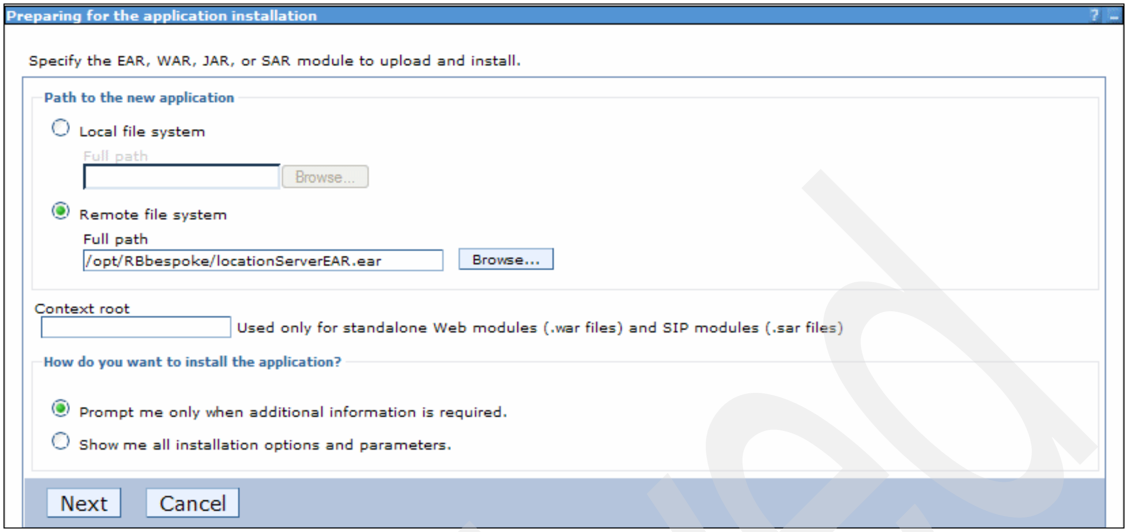


Figure 13-3 Preparing for the Application Installation

9. In the Select Install Options window, accept the defaults, and then click **Next**.
10. In the Map Modules to Server window, select Location, and then click **Next**.
11. In the Summary window, verify the installation options.
12. Click **Finish**.
13. In the Installation Status window, click **Save to Master configuration**.
14. Once the configuration has been saved, the application must be started.
    a. Click **Applications** → **Enterprise Applications**.
    b. In the Enterprise Applications window, select Location Server.
    c. Click **Start**.

Configure the simulated Location data

1. Copy the file terminallocation.db to the root directory on the Linux test server. Example 13-3 shows the listing of the location data in terminallocation.db file.
2. Edit termininallocation.db and change <wtl_ipaddress> for each SIP URL to the IP address where the SIP softphone for that technician is installed. If you have used a similar configuration to the one for this sample environment, then this is the IP address for the Windows client test machine.

Example 13-3 Contents of simulated device locations

```
# This file contains the terminal locations, fields are # separated
# lines beginning with a # are treated as comments
```

```
# The field order is URI, latitude, longitude, altitude, accuracy
#List of technicians locations
sip:bruno@<wtl_ipaddress>:5999#89.89#12.14#1278#1
sip:callum@<wtl_ipaddress>:5998#89.89#14.12#1278#1
sip:cameron@<wtl_ipaddress>:5060#89.89#12.12#1278#1
sip:jochen@<wtl_ipaddress>:5095#89.89#12.15#1278#1
sip:philippe@<wtl_ipaddress>:5997#89.89#13.12#1278#1
sip:rebecca@<wtl_ipaddress>:5996#89.89#12.15#1278#1
#simulated FindHelp Caller location
sip:caller@<wtl_ipaddress>:5070#89.89#12.13#1278#1
```

13.2.3 Application deployment

The following applications must be deployed for executing the IMS use cases for the FindHelp sample service.

FindHelp BPEL Flow

The following instructions explain how to deploy the FindHelp BPEL flow to the WebSphere Process Server running on the Windows test machine.

The BPEL flow is included in the file FindHelp_BPEL.zip which is part of the additional materials for this redbook (see Appendix C, “Additional material” on page 637).

> **Attention:** Alternatively you may develop your own FindHelp BPEL flow by following the instructions given in 12.3, “BPEL development” on page 361.

1. Download the file FindHelp_BPEL.zip to the Windows test machine.
2. Start WebSphere Integration Developer and open the Business Integration perspective.
3. Import the file FindHelp_BPEL.zip from the local test machine disk to the WebSphere Integration Developer workspace.
4. Select **File** → **Import** → **Project Interchange**.
5. Click **Next**.

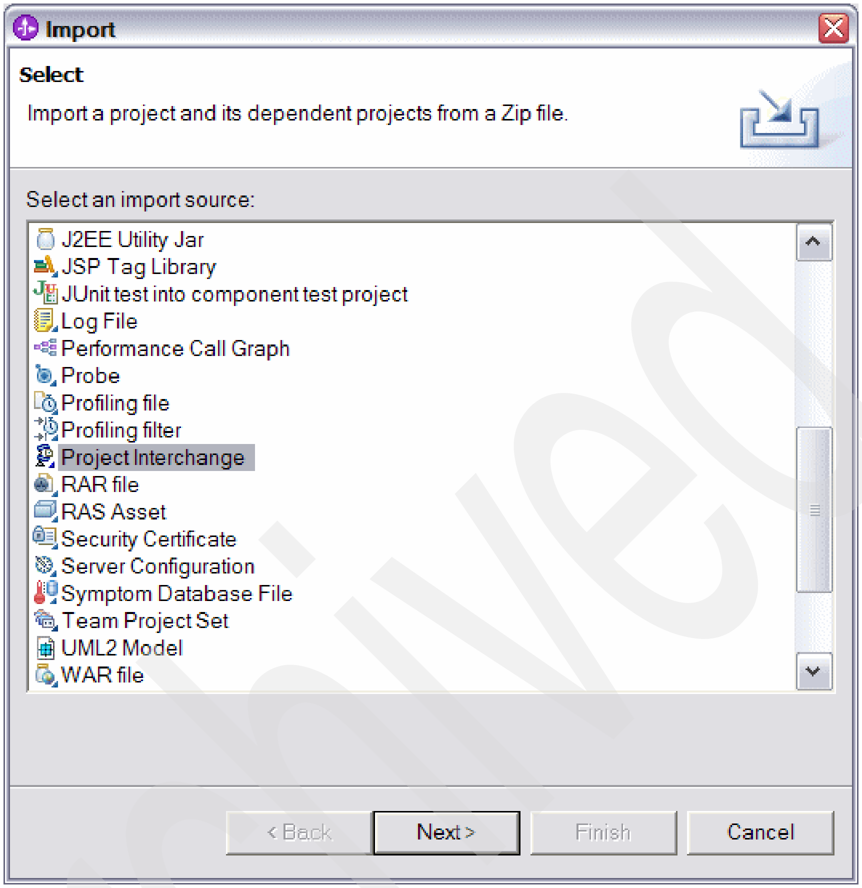


Figure 13-4 Import FindHelp BPEL Flow

6. Navigate to the directory which contains the file FindHelp_BPEL.
7. Select the file, and click **Open**.
8. In the **Import Projects** window, click **Select All**.
9. Click **Finish**.
10. Open the FindHelp Assembly diagram.
11. Click **+** in the **Business Integration** window to expand the FindHelp business process.
12. Select **FindHelp** → **FindHelp**.
13. Double-click **setupcall**.

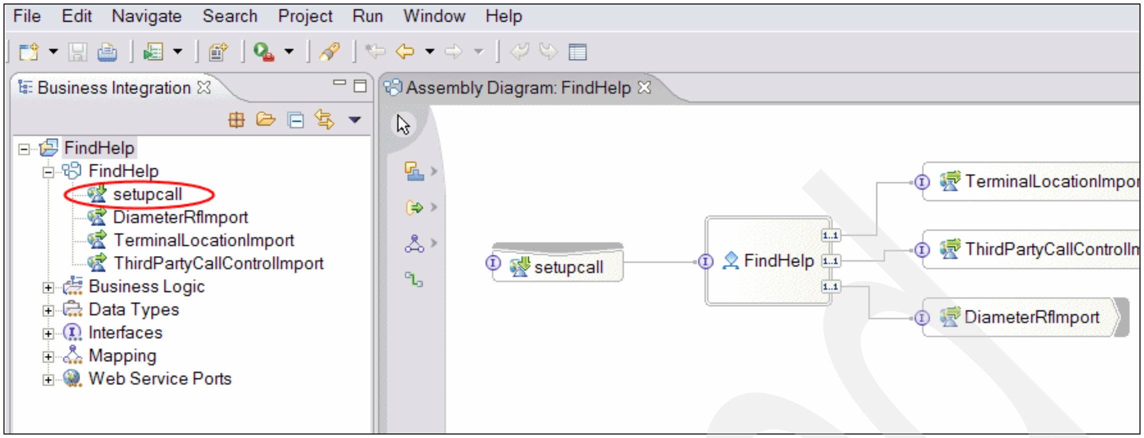


Figure 13-5 Open the Assembly Diagram: FindHelp

14. Click the Interface icon which is on the left side of the `TerminalLocationImport`.

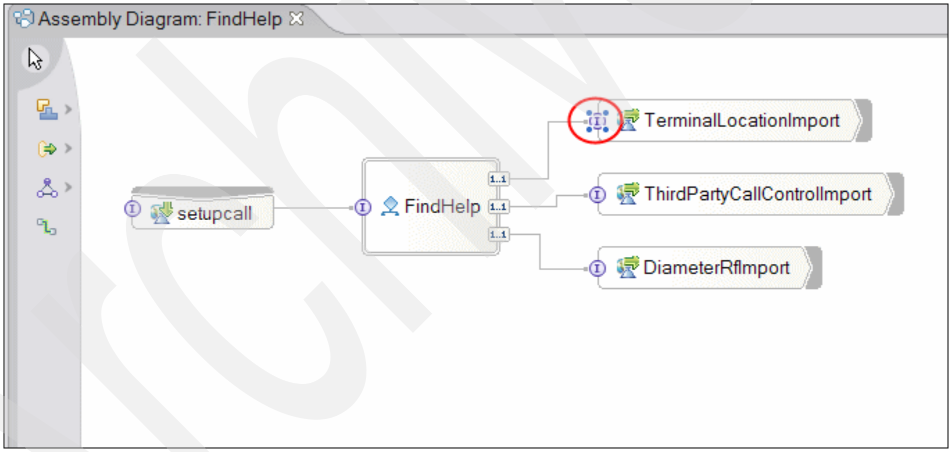


Figure 13-6 Click the Interface Icon for TerminalLocationImport

15. Click **Binding** in the Import: TerminalLocationImport Properties window as shown in Figure 13-7 on page 453.

    Change the IP address in **End point** to the IP address of the server where you have installed the LocationServerEar. If you are following the installation used in this redbook, this is the IP address of the Linux test server, <lts-ipadress>. Leave `:9080` as the port number.

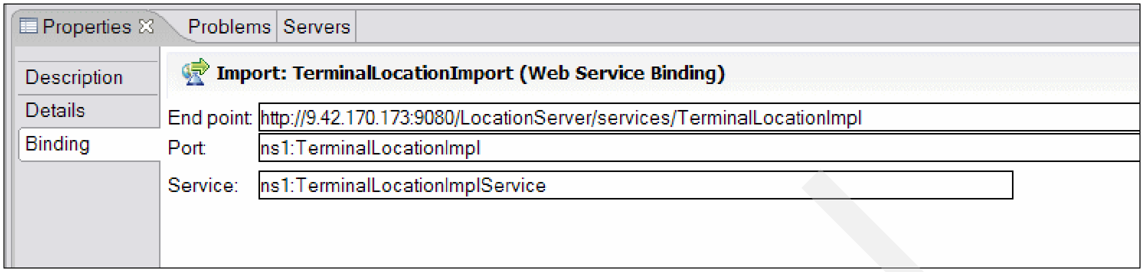


Figure 13-7 Binding for TerminalLocationImport

16.Click the Interface icon which is on the left side of the `ThirdPartyCallControlImport`.

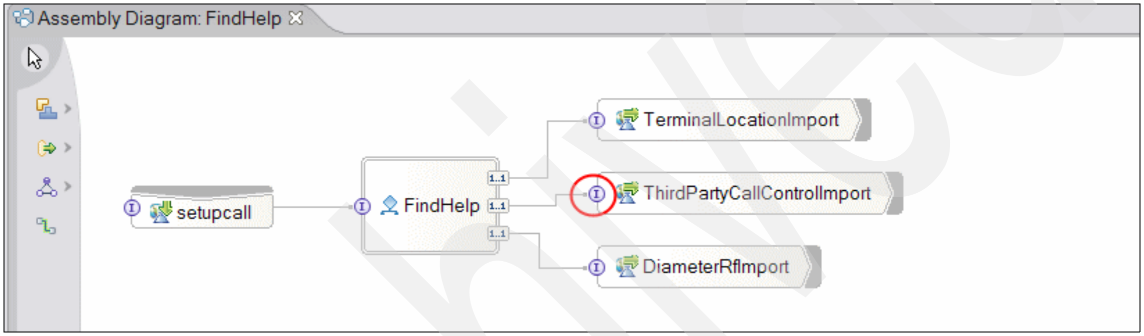


Figure 13-8 Click Interface icon for ThirdPartyCallControlImport

17.Click **Binding** in the Import: ThirdPartyCallControlImport Properties window as shown in Figure 13-9.

Change the IP address in **End point** to the IP address of the server where you have installed the ThirdPartyCallControl.ear. If you are following the installation used in this redbook, this is the IP address of the Linux test server, <lts-ipadress>. Leave **:**`9084` as the port number.

Figure 13-9 Binding for ThirdPartyCallControlImport

18.Click the Interface icon which is on the left side of the `DiameterRfImport`.

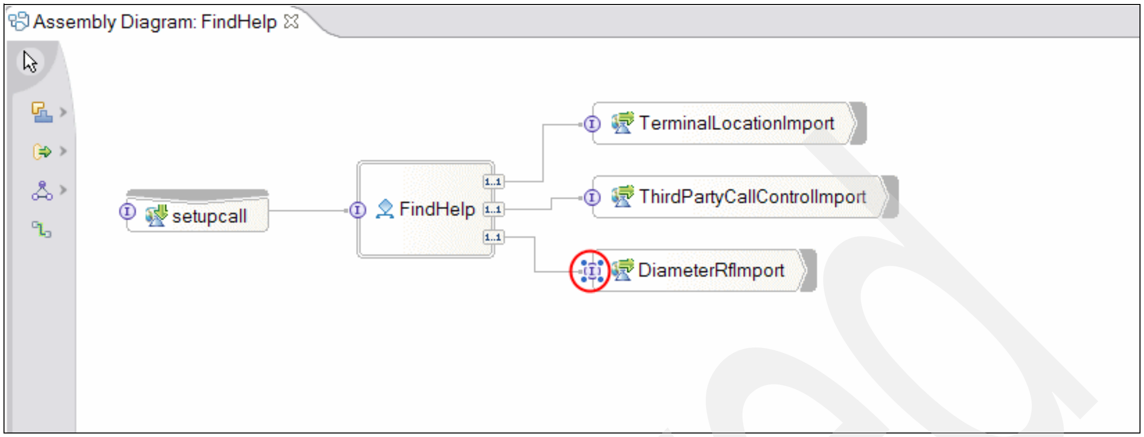


Figure 13-10 Click the Interface icon for DiameterRfImport

19..Click **Binding** in the Import: DiameterRfImport Properties window as shown in Figure 13-11.

Change the IP address in **End point** to the IP address of the server where you have installed the Diameter Web service. If you are following the installation used in this redbook, this is the IP address of the Linux test server, <lts-ipadress>. Leave **:**`9083` as the port number.

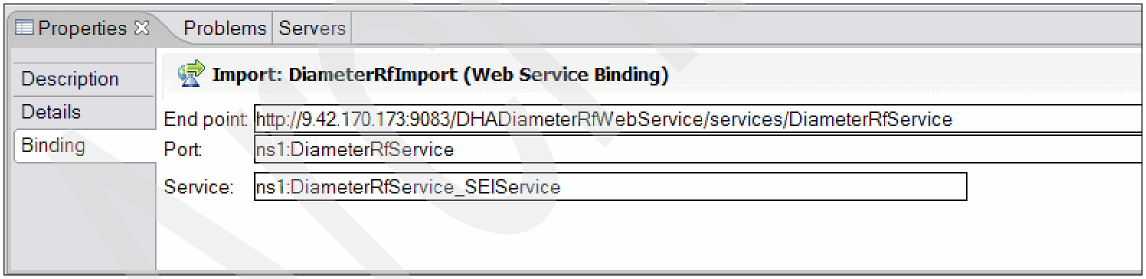


Figure 13-11 Binding for DiameterRfImport

20.To start the WebSphere Process Server, select **Project** → **Clean**.

21.Select **Clean all projects**.

22.Click **OK**.

23.Click the **Servers** window and select **WebSphere Process Server v6.0**.

24. Start the WebSphere Process Server by right-clicking and selecting **Start**.

    Observe the server start progress in the **Console** window. This may take a few minutes. The server is successfully started when you receive the message `ADMU3000I: Server server1 is open for e-business; process id is nnnn`

25. When the WebSphere Process Server has successfully started:

    a. Right-click **WebSphere Process Server v6.0** in the Servers window.

    b. Select **Add and Remove Projects**.

26. In the **Add and Remove Projects** window:

    a. Select **FindHelpApp** under **Available projects**.

    b. Click **Add**.

27. Click **Finish**.

SIP Servlet

The following instructions explain how to install and deploy the FindHelp SIP Servlet to the WebSphere AS 6.1 installed on the Linux Test Server.

1. Copy the file **FindHelp.sar** to a development machine where WebSphere AST V6.1 is installed. This is the Windows test machine in this test environment.
2. Open the WebSphere Application Server Toolkit V6.1.

   Select **IBM WebSphere** → **Application Server Toolkit V6.1** → **Application Server Toolkit** from the Windows Start command list.
3. Import the **FindHelp.sar** file into WebSphere AST.
4. Select **File** → **Import** → **SAR file**.
5. Click **Next**.
6. Click **Browse** to navigate to the location of the **FindHelp.sar** file.
7. Click **Open**.

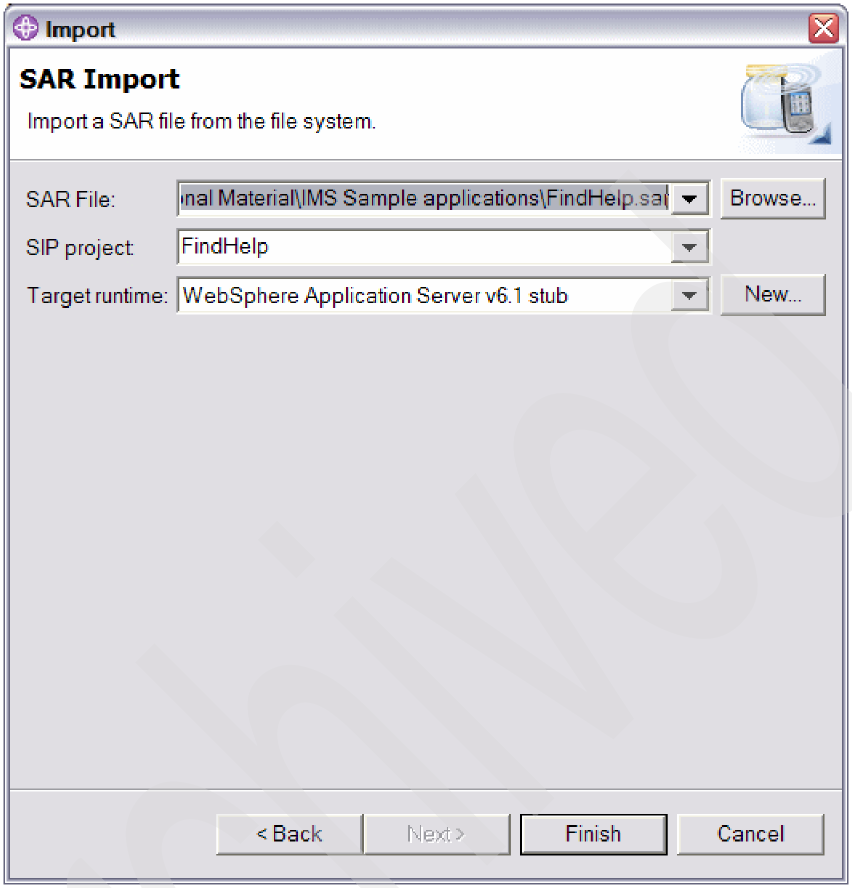


Figure 13-12 Import SAR file

8. Click **Finish**.
9. In the Project Explorer window, double-click **Other Projects** → **FindHelp** → **SIP Deployment Descriptor**.
10. Within the **SIP Deployment Descriptor** window, click the Source tab.

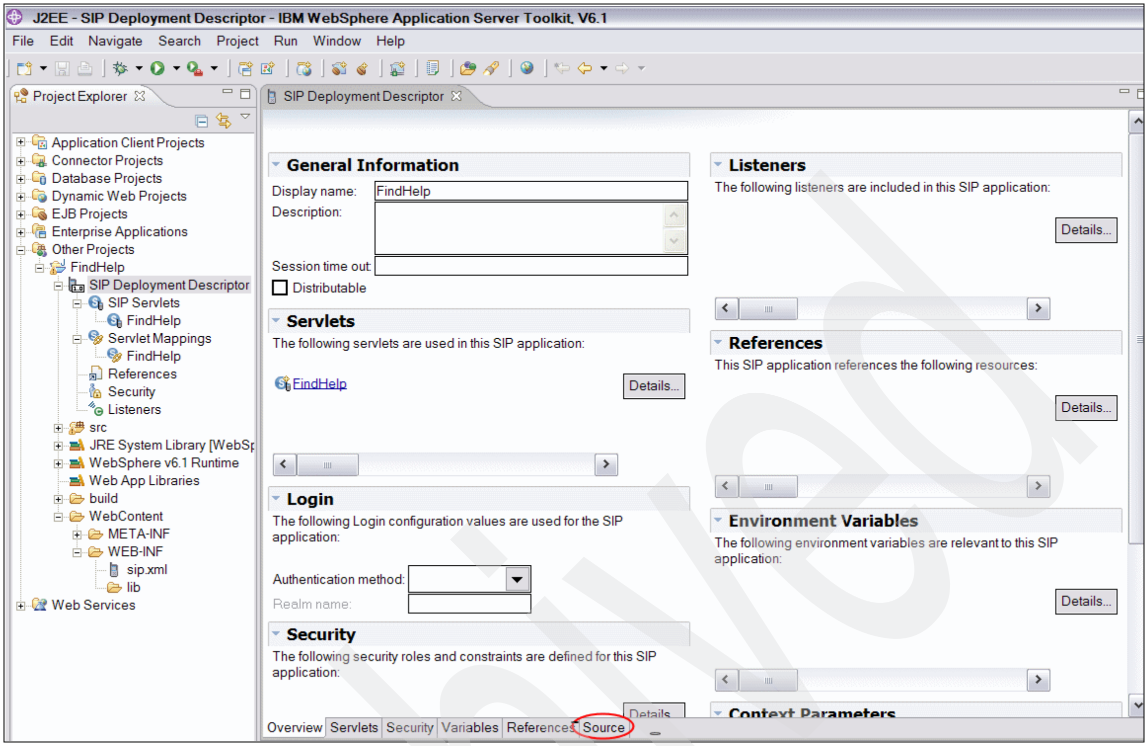


Figure 13-13 Select SIP Deployment Descriptor Source

11. Modify **Presence Server Hostname** to the IP address of the server where the Presence Server is installed.
    a. In the configuration for this sample test environment, this is the IP address of the Linux test server, <lts_ipaddress>.
    b. If you have use the same configuration as in this sample test environment, the PresenceServerPortNumber should be :5063.
    c. Modify the ProcessServerURI to the IP address of the machine where the WebSphere Process Server is installed, this is the Windows test machine IP address <wtl_ipaddress> in this sample test environment.
12. Select **File** → **Save** to save your changes to the servlet.
13. Export **FindHelp.sar** by selecting **File** → **Export**.
14. Select **SAR File**.
15. Click **Next**.

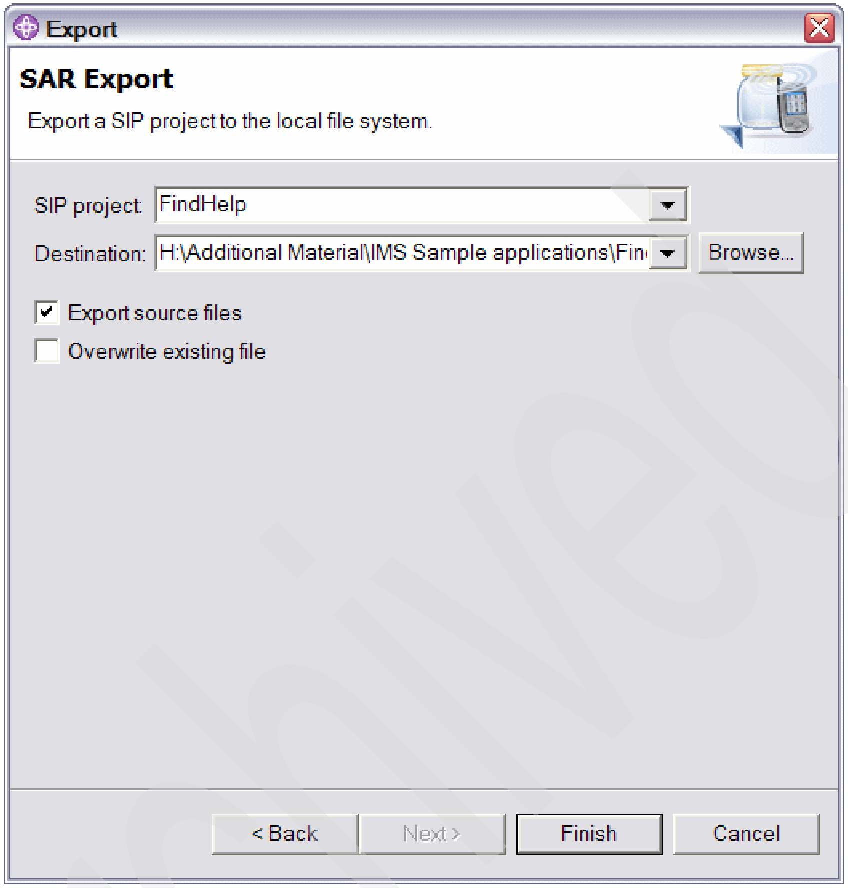


Figure 13-14 Export modified FindHelp.sar

We are now ready to deploy the FindHelp SIP Servlet to the Linux test server, this is on Server Node 1 for this test environment.

1. Open the **Integration Solution Console** for the **WebSphere Application Server Node 1**.
2. Click **Applications**.
3. Click **Install New Applications**.
4. Select the radio button **Remote File System**.
5. Click **Browse**.
6. Navigate to the directory where you have exported the FindHfile /root/opt/RBbespoke/ and select the radio button for **FindHelp.sar**.
7. Enter `FindHelp` in **Context Root**.

8. Click **Next**.

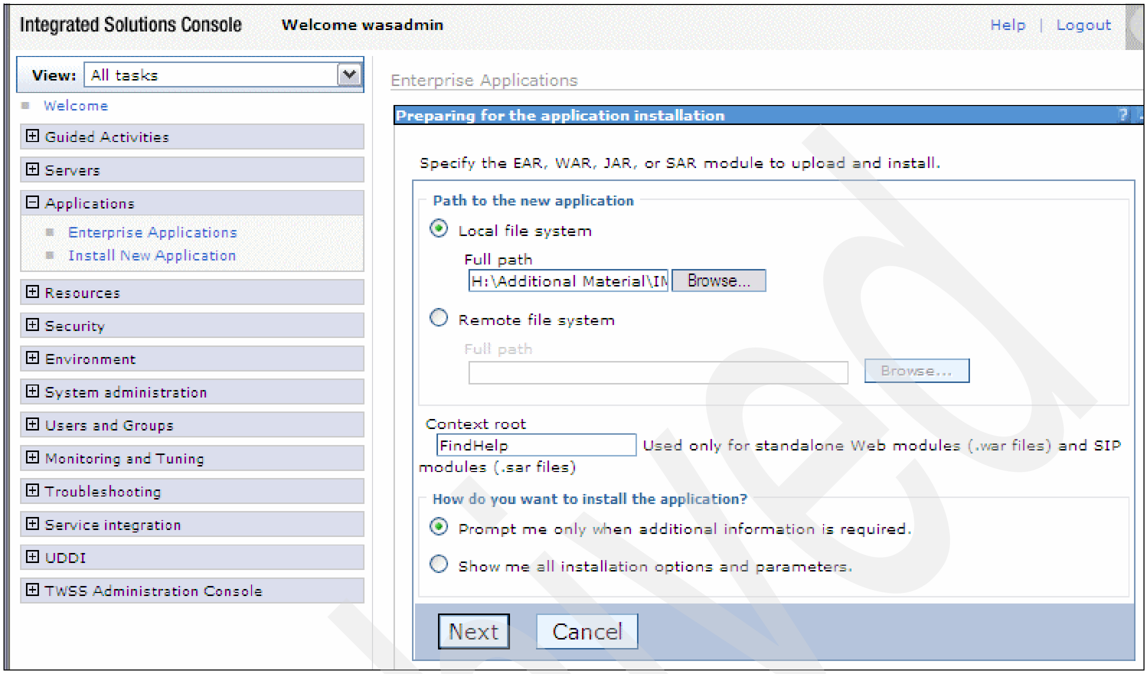


Figure 13-15 Specify SAR file to upload and install

9. Accept defaults for **Installation Options**.
10. Click **Next**.
11. In the **Map Modules to Servers** window, click **Next** and verify the installation summary.
12. Click **Finish**.
13. You should receive the message `Application FindHelp_sar installed successfully`.
14. Click **Save directly to the master configuration**.
15. Start the FindHelp SIP Servlet.

13.2.4 Device client setup

The Device client used by the technicians and the caller is simulated using freeware which can be downloaded from Internet.

- **SIPp**: Is used to implement the SIP dialog support for the FindHelp application on the client side.
- **Softphones**: sipXphone and SJPhone are two SIP freeware softphone. These softphones were used for handling Third Party Call Control (3PCC) in the FindHelp scenario between the Caller and Tech1.

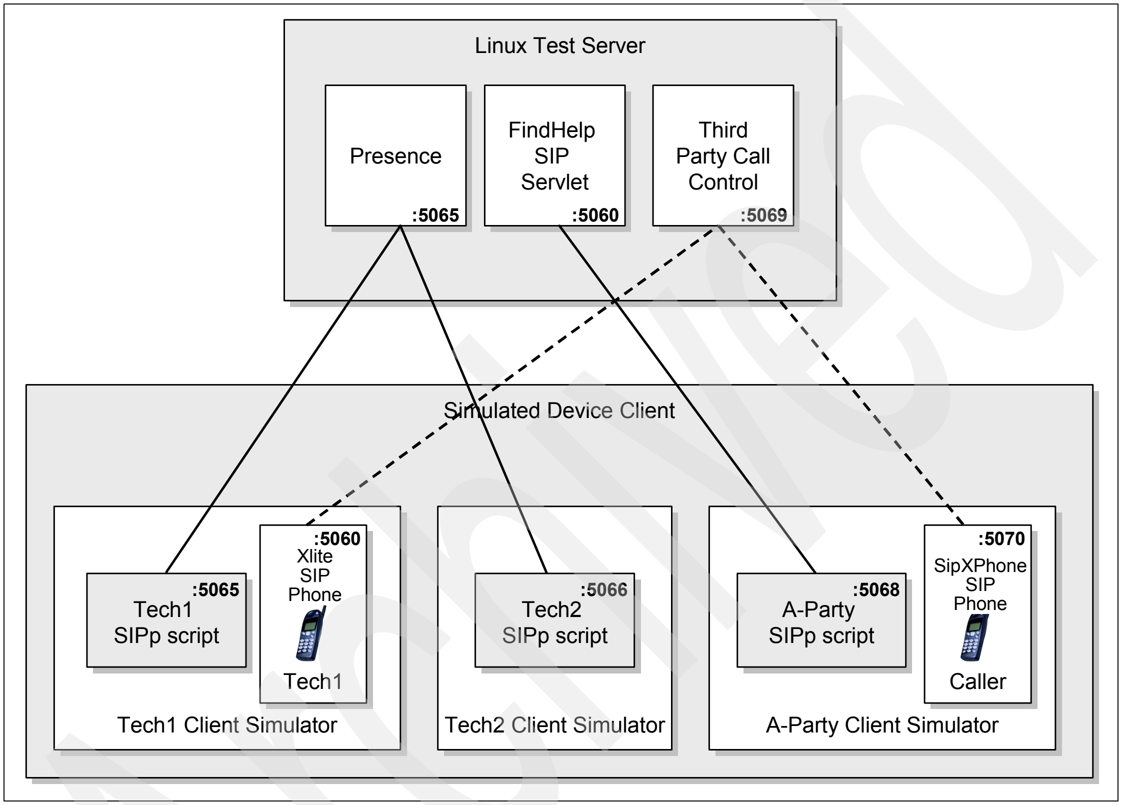


Figure 13-16 Device Client Setup

> **Note:** You can download SIPp from `http://sipp.sourceforge.net`. The version we used for this redbook is sipp-unstable 1.1rc4. The Windows .exe file is sipp-1.1rc4.win32-setup.exe.

SIPp installation

1. Download the SIPp package for **Windows** platform.
2. Run the SIPp setup wizard.

Note: Depending on whether the software is already installed or not on your PC, you may have to install cygwin (it is likely to be the case if you encounter errors running SIPp scripts, such as “error opening display...”). Cygwin is a Linux-like environment for Windows. It consists essentially of a DLL (cygwin1.dll) which acts as a Linux API emulation layer providing substantial Linux API functionality. It also includes a collection of useful tools.

If you need to install cygwin, you can download it from www.cygwin.com. Install the latest version (Version 1.5.19-4 was used for our sample test environment). Choose default options when downloading cycwin components.

Verify the SIPp installation

Once you have installed SIPp, you can invoke it by starting a SIPp shell:

1. Click **Start** → **Programs** → **SIPp** → **Start SIPp shell**.
2. A DOS window should appear as shown in Figure 13-17.

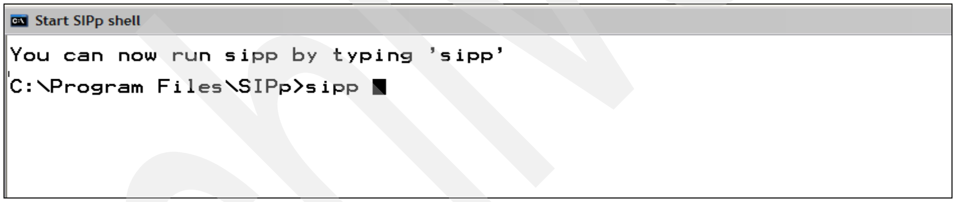


Figure 13-17 Starting a SIPp shell

sipXphone setup

1. Start the sipXphone by selecting **SIPFoundry** → **sipXphone** → **sipXphone** from the Windows **Start** → **Program** list.
2. Verify that the Administrator Web Interface is activated.

Note: See A.2, “SIP device client installation” on page 502 for instructions for installing sipXphone.

3. Open the SipXPhone Administration Console by pointing your Web Browser to:

   `//localhost:80`

4. Enter `admin` in **User name**.
5. Click **OK**.

6. Click **Administration** → **Phone Configuration**.
7. Under **SIP Servers**
   a. Change **(SIP_TCP_PORT**) and **(SIP_UDP_PORT)** to use port `5070`.
   b. Scroll to the bottom of the window.
8. Click **Save**.

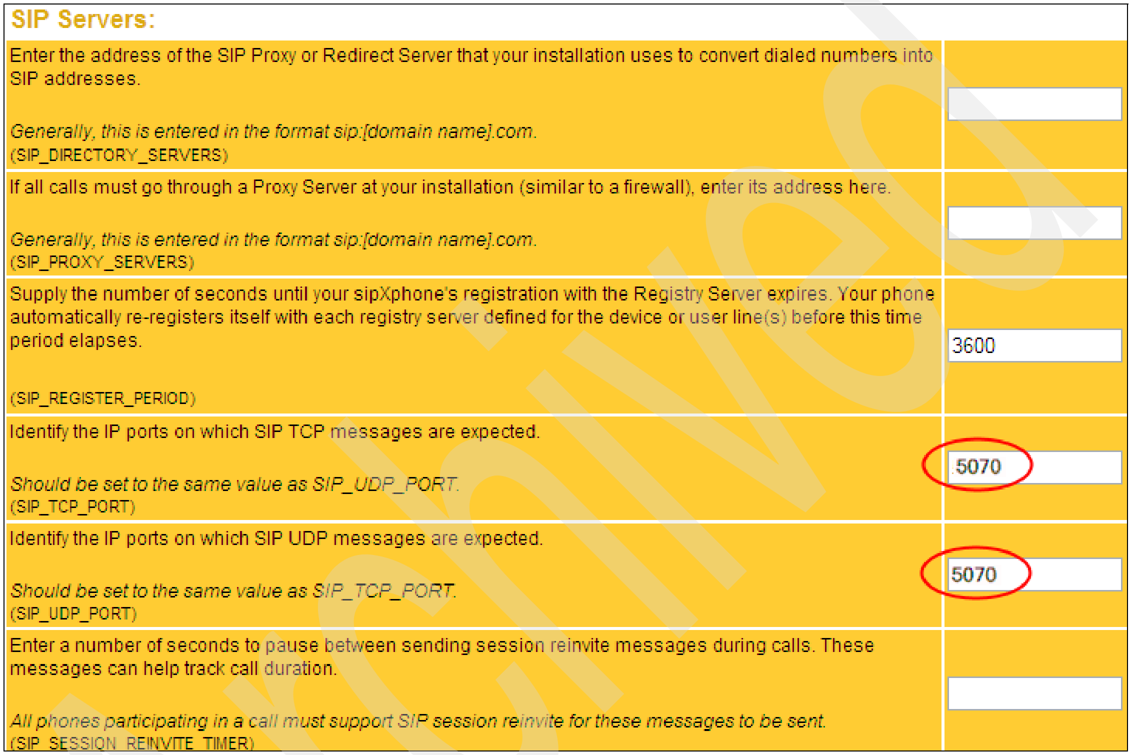


Figure 13-18 Configure SIP ports for sipXphone

9. Click **Preferences** → **Lines**.
10. In the Line Preferences window under Line Edit, click **Edit** for the device line.

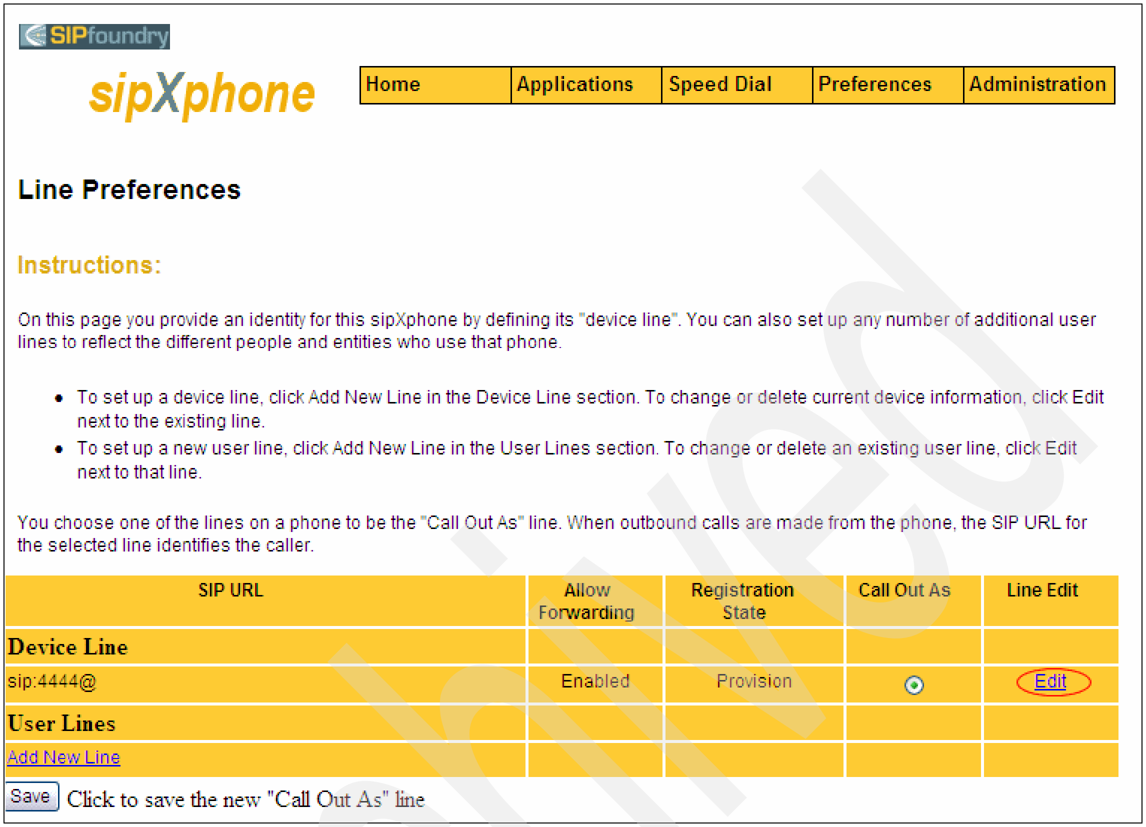


Figure 13-19 Edit Device Line for sipXphone

11. Replace the contents of **SIP URL:** with `sip:caller@<wtl_ipaddress>:5070`.

> **Note:** `<wtl_ipaddress>` is the IP address of the Windows test machine

12. Click **Update**.
13. Click **Save**.

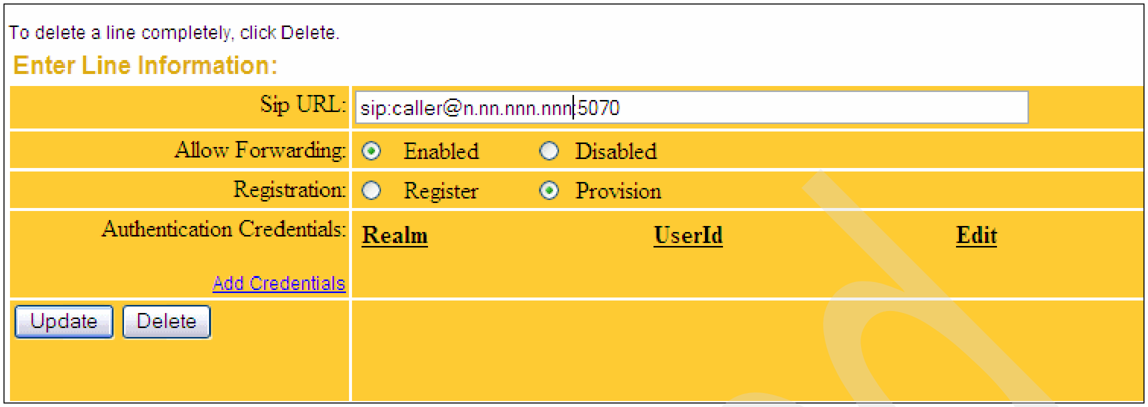


Figure 13-20 Enter SIP URL for Caller SIP phone

14. Select **Administration** → **Phone Configuration**.
15. Click **Restart**.
16. Click **OK**.
17. Switch to the sipXphone.
18. Click **OK** to verify that you want to restart.

SJphone

1. Start the SJPhone by selecting **SJphone** → **SJphone** from the Windows **Start** → **Program** list.
2. Right-click the SJphone, and select **Options**.
3. In the **User Information** window, enter `Tech1` in the Name field.
4. Click **OK**.

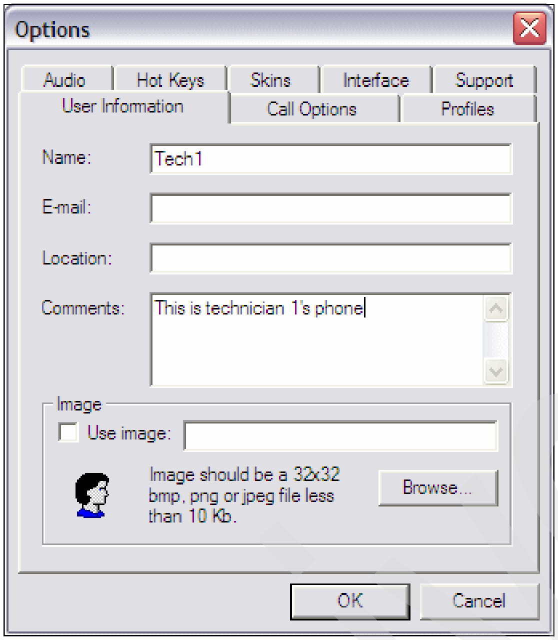


Figure 13-21 Add User Information

13.2.5 Installing the IBM Diameter CCF Simulator

To run the sample application you need to install a standalone Java application that simulates a Charging Collection Function (CCF) server.

You can obtain this application by downloading the additional materials for this redbook available in Appendix C, "Additional material" on page 637.

To install the application:

1. Unzip the DiameterTestServer package to any directory.
2. Switch to the unzip directory.
3. Run the Diameter simulator:

```
java_bin_root/java -classpath com.ibm.ws.diameter_6.1.0
.jar:TestServer.jar com/ibm/diameter/test/RfdiameterListener
port_number localHostname localHostIpAddress
```

   Where:

   – port_number

     Is the port number to be used for the Diameter protocol. It is `3868` by default but if you have installed the simulator and the Rf accounting Web

Service on the same machine it must be set to a different value that matches the `con1.remotePeerPort` field in the properties file.

– localHostname

  Is the server hostname that runs the simulator

– localHostIpAddress

  Is the server IP address that runs the simulator.

Figure 13-22 shows the messages produced by the simulator when it is successfully started.

```
[root@kpmgyw0 DiameterTestSimulator]# /opt/IBM/WebSphere/AppServer/java/bin/java -classpath com.ibm.ws.diameter_6.1.0.jar:Tes
tServer.jar com/ibm/diameter/test/RfDiameterListener 3868 kpmgyw0.itso.ral.ibm.com 9.42.171.117

Usage: RfDiameterListener port originHostName hostIpAddress

** Setting up vsavpdict
Done configuring vsapvdict
Using originHostName: kpmgyw0.itso.ral.ibm.com
Using hostIpAddress: 9.42.171.117
Waiting for a client - modified
We have a client
Received this packet: [Command Code: 257 (CER)][Command Flags: 0x80][Msg Length: 200][HopByHop: 0xF54AA74A][EndToEnd: 0xD42FD
ODA][Version: 1][Origin-Host=kofrzoy.itso.ral.ibm.com][Origin-Realm=itso.ral.ibm.com][Host-IP-Address=[B@22c622c6][Vendor-Id=
2][Product-Name=IBM WebSphere Diameter Component][Supported-Vendor-Id=10415][Inband-Security-Id=0][Vendor-Specific-Applicatio
n-Id=[[Vendor-Id=10415], [Acct-Application-Id=16777218]]]
Received a CER - creating response
No error
Sent packet
```

Figure 13-22 Starting the Diameter CCF simulator

> **Note:** The Capability Exchange (CER) messages are messages that are periodically exchanged between the Diameter client and the server.

13.3 Executing the test scenarios

To exercise the test environment, we execute the use cases we created for the FindHelp sample IMS application:

- Administrator adds service topic
- Technician's publishing of presence information
- Caller request for FindHelp service topic LockOut

The step-by-step description for running the use cases is provided in the sections that follow.

13.3.1 Use Case 1: Administrator adds service topic

The FindHelp Service Administrator maintains service topics and associates the technicians to the appropriate service topic. We have already seen in13.2.1, “Group List Server setup” on page 445 how you can create the resource-list and RLS services documents in XML and store them into the Group List Server using XCAP protocol. The Group List Server also offers a Web user interface to administer the groups and users.

In this use case, the Administrator creates another service topic, PestControl by adding another group called PestControl using the GLS Web user interface.

1. Logon to the GLS Console by pointing your Web browser to:
   <lts_ipaddress>:9081/GLSAdmin/GLMAdmin

   Login with GLSUser1 and the appropriate password.
2. Click **Group** → **Create**.
3. Enter the following data:
   a. PestControl in **Group Name**.
   b. pest.control in **Group Domain**.
4. Click **Create**.

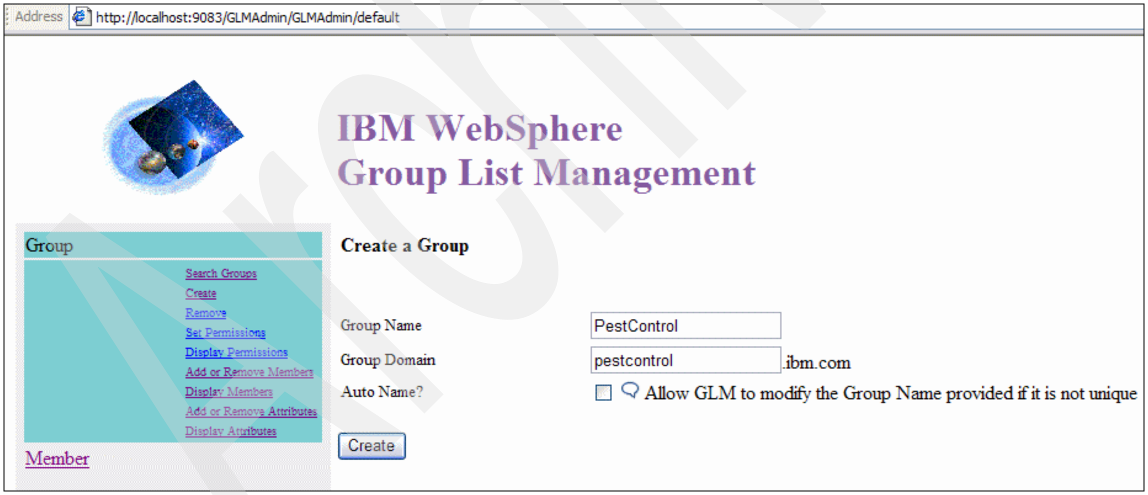


Figure 13-23 Create a Group: PestControl

Next, we add technicians that specialize in pest control. This new resource list will allow the FindHelp service to identify the technician’s location and availability which specialize in pest control.

We assume that we have two multi-faceted technicians that are skilled in both locksmithing and pest control, Tech3 and Tech4.

1. On the Group List Management window, click **Group** → **Add or Remove Members**.

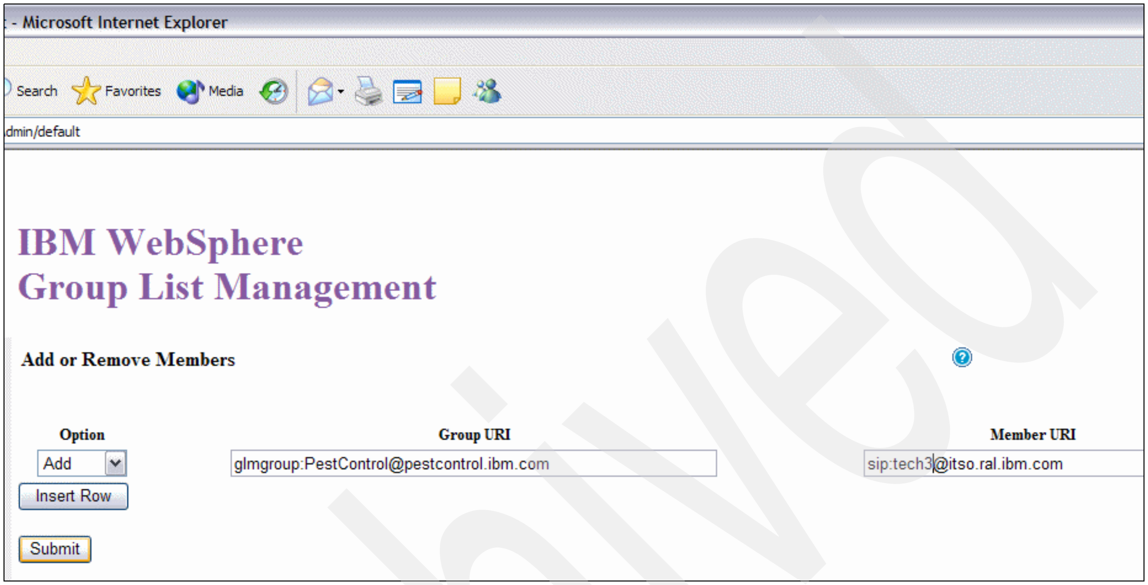


Figure 13-24 Add members to PestControl Group

2. Enter `sip:tech3@itso.ral.ibm.com` in **Member URI**.
3. Click **Submit**.
4. Click **Insert Row**.
5. Enter `sip:tech4@itso.ral.ibm.com` in **Member URI**.
6. Click **Submit**.

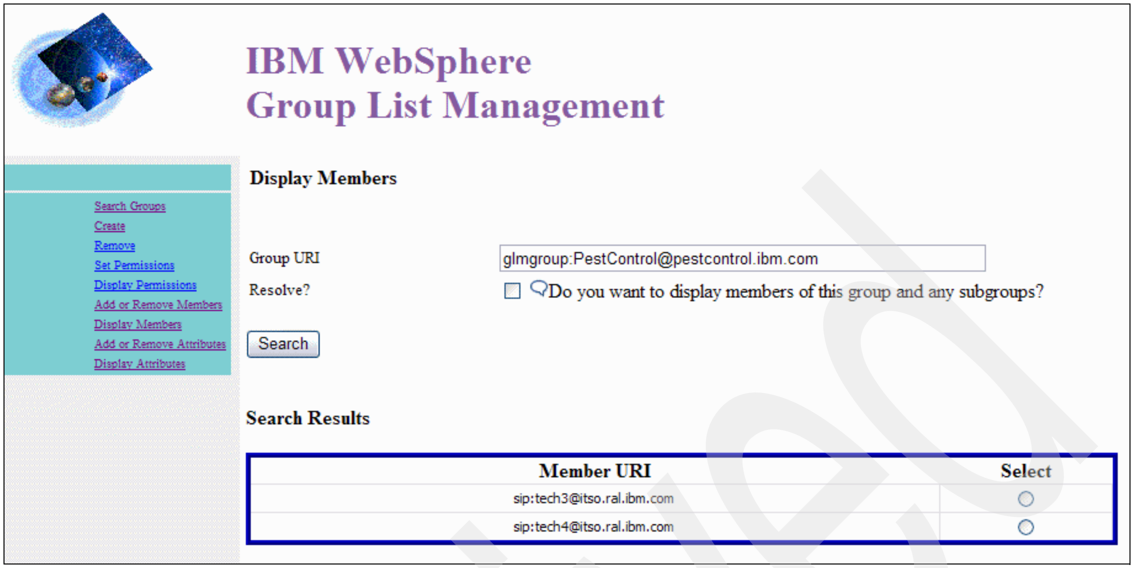


Figure 13-25   Display members of the PestControl group

You have now created a group PestControl containing two members: tech3 and tech4. This group is associated with a SIP URL that is used by the service FindHelp to identify which technicians belong to the PestControl group and are therefore skilled in PestControl.

13.3.2 Use Case 2: Publish Technician Status

The lockout technicians publish their availability and presence status. For this test case scenario, the SIP control traffic is being simulated by SIPp using XML scripts. We provide two scripts which simulate publishing two of the technicians but you may duplicate and change the scripts to simulate more technicians if desired.

Publish_technician1.xml and publish_technician2.xml are SIPp scripts that publish the presence information for Tech1 and Tech2 users, respectively. These scripts simply perform a SIP_PUBLISH or submit presence information to the presence server.

1. Edit the publish_technician1.xml file.
2. Change <wtl_ipaddress> to the IP address of the Windows test machine.
3. Change <tech1_phone_port> to port which the SIP softphone for Tech1 is listening.

Example 13-4 Edit publish_technician1.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">

<!-- ############################################################ -->
<!-- Objective:                    -->
<!--                        -->
<!--Publish_Tech1       -->
<!--                        -->
<!--This script sends a PUBLISH request for Tech1             -->
<!--The following strings must be changed before executing    -->
<!--    this script:  -->
<!--    <wtl_ipaddress> - ip-address of the test machine with the   -->
<!--                    SIP softphone for Tech1 installed        -->
<!--    <tech1_phone_port> - port which the SIP softphone for     -->
<!--                  Tech1 is listening                      -->
<!-- Date:                                                        -->
<!--    29-June-2006                                              -->
<!--                                                              -->
<!-- ############################################################ -->

<scenario name="Publish_tech1">


  <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
  <!-- Publish first  user from event list  -->
  <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->

 <send>
    <![CDATA[

      PUBLISH sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Max-Forwards: 70
      From: <sip:cameron@itso.ral.ibm.com>;tag=[call_number]
      To: <sip:cameron@itso.ral.ibm.com>
      Content-Type: application/pidf+xml
      Via: SIP/2.0/[transport] [local_ip]:[local_port]
      CSeq: 1 PUBLISH
      Call-ID: [call_id]
      Expires: 60
      Event: presence
      Content-Length:[len]

     <?xml version="1.0" encoding="UTF-8"?>
     <presence entity="cameron@itso.ral.ibm.com"
xmlns="urn:ietf:params:xml:ns:pidf">
        <tuple id="1234560001">
                <status>
```

```
                <basic>open</basic>
             </status>
          <contact>sip:cameron@<wtl_ipaddress>:<tech1_phone_port></contact>
     </tuple>
 </presence>

]]>
</send>

<recv response="200" optional="false">
</recv>


</scenario>
```

4. Repeat Step 2 and 3 for Publish_Technician2.xml.

We are now ready to execute the scripts.

5. Open a DOS command window and navigate to the directory where SIPp is installed.
6. Type the command in Example 13-5.
   a. Substitute <rb_scr_dir> with the directory where you have stored the modified Publish_Technician1.xml script.
   b. Substitute <lts_ipaddress> with the IP address of the Linux test server where the Presence server is installed.
   c. Substitute <ps_port> with the TCP/UDP port number the Presence Server is using for receiving SIP messages which in this case is port `:5065`.

> **Note:** Tech1 is using port :5065 and Tech2 is using port :5066 for SIPp on the Windows test machine in this sample environment configuration. The SIP softphone for Tech1 is using port :5060. The SIP softphone for caller is using port :5070.
>
> The SIP softphone for Tech2 was not installed as this would require a third SIP softphone. If desired, you may download a third (different) SIP softphone or alternatively install the third SIP softphone on another test machine configured with a UDP port available on that machine (for example :5060).

Example 13-5 Invoke SIPp with Publish_Tech1.xml

```
sipp -p 5065 -sf <rb_scr_dir>Publish_Tech1.xml" -trace_msg -m 1
<lts_ipaddress>:<ps_port>
```

The DOS command window will appear similar to Figure 13-26 on page 472.

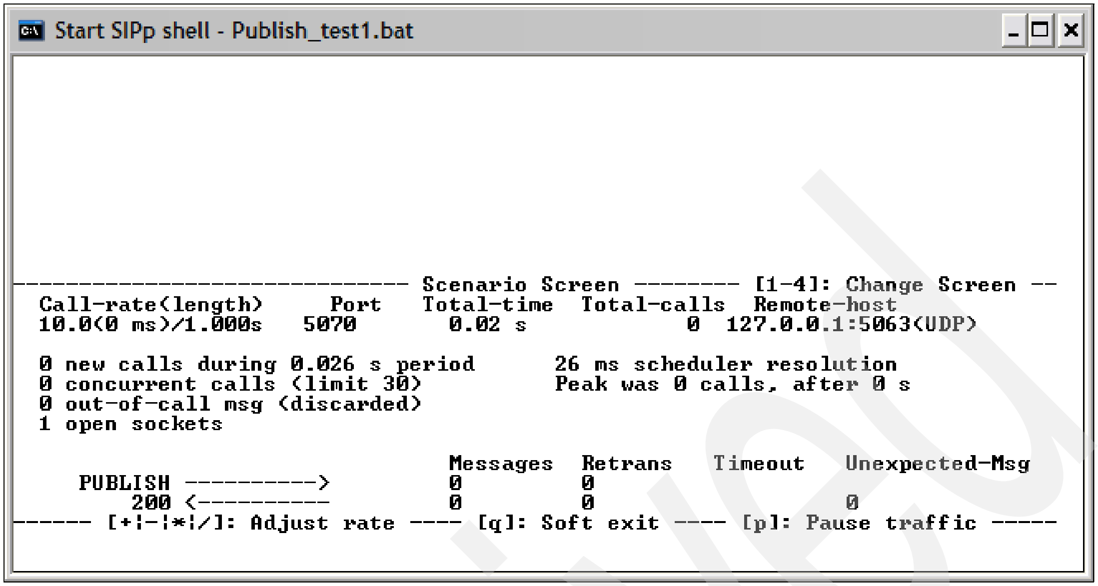


Figure 13-26 SIPp result for publish_tech1.xml

Next, we publish Tech2's status.

1. Open a second DOS command window and navigate to the directory where SIPp is installed.
2. Type the command in Example 13-6.
   a. Substitute <rb_scr_dir> with the directory where you have stored the modified Publish_Technician2.xml script.
   b. Substitute <lts_ipaddress> with the IP address of the Linux test server where the Presence server is installed.
   c. Substitute <ps_port> with the TCP/UDP port number the Presence Server is using for receiving SIP messages which in this case is port :5065.

Example 13-6 Invoke SIPp with Publish_Tech2.xml

```
sipp -p 5066 -sf <rb_scr_dir>Publish_Tech2.xml" -trace_msg -m 1
<lts_ipaddress>:<ps_port>
```

The DOS command window will appear similar to Figure 13-27 on page 473.

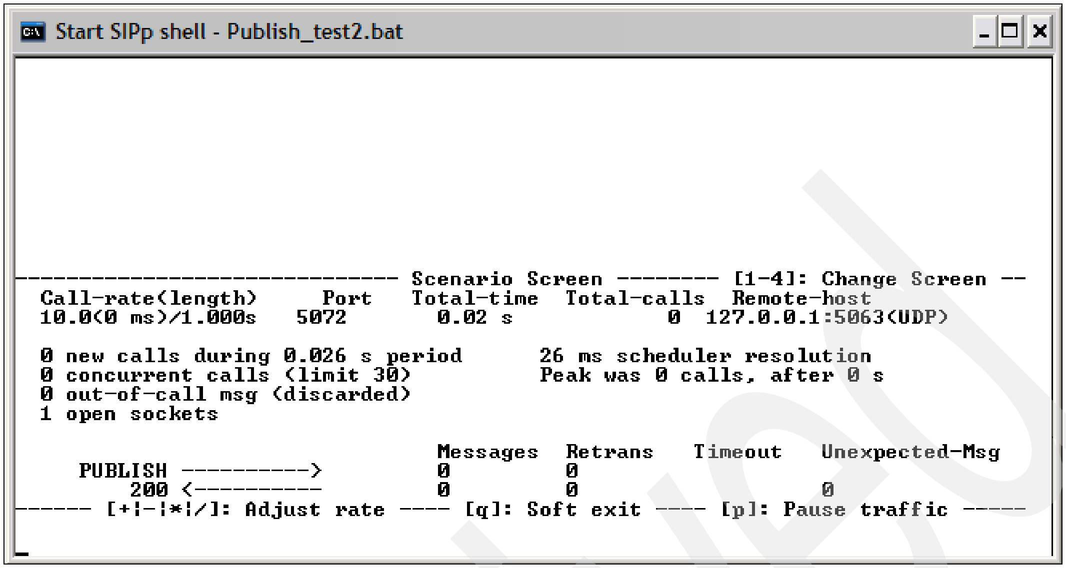


Figure 13-27 SIPp result for publish_tech2.xml

> **Note:** the SIP PUBLISH message contains the following important information:
>
> - The target URI is the SIP URI of the Presence Server.
> - The “Expires” header indicates the lifetime of the event state publication.
> - As specified in RFC3903, an initial PUBLISH request MUST NOT contain a “SIP-If-Match” header. However, an Event Publication Agent (EPA) must use this header in order to modify, remove or remove a previously published state.
> - The body of the PUBLISH request carries the published event state in the form of a PIDF presence format XML document.

13.3.3 Use Case 3: Caller requests FindHelp Service

The caller has locked himself out of his car and needs a locksmith. He is not sure where exactly he is located but he has a appointment in the next hour and needs a lockout service immediately. He invokes the FindHelp service with the topic Lockout. The FindHelp service finds the closest, available lockout technician and establishes a call between the caller and the lockout technician.

The SIPp XML scripts in 13.3.2, “Use Case 2: Publish Technician Status” on page 469 simulate the technician’s devices. For this use case, we also provide a SIPp XML script file which simulates the caller invoking the FindHelp service.

Caller_requests_FindHelp.xml is the SIPp script that implements the SIP dialog for FindHelp application. This script starts a SIP dialog between the caller and the FindHelp application, using SIP_INVITE command, waits for “100” and “200” RC, issues a SIP ACK command to acknowledge the reception of “200” RC, then waits for a SIP INFO message containing the name and phone number of the technician for which a call is being established, displays that information on the screen using `dos MSG` command, and finally waits for the reception of a BYE command that terminates the session.

> **Note:** The use of a CSCF is outside of the scope of this use case. The use case assumes that the user has already registered. All SIP traffic from the simulated device client is directly routed to the SIP Serlvet.

Setup the Caller_requests_Findhelp script

1. Edit the file Caller_requests_Findhelp.xml.
2. Change <wtl_ipaddress> to the IP address of the Windows test machine.
3. Change <windows_userid> to the Windows user ID logged on in the Windows test machine. This will allow a pop-up window to be displayed upon successful completion of the SIPp script.
4. Change <caller_phone_port> to the port used by the SIP softphone for the caller. This value will be used to setup the Third-Party call between the caller and the technician. In this sample scenario, the sipXphone is listening on port :5070.

Example 13-7 Edit Caller_requests_FindHelp SIPp script

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- ############################################################ -->
<!-- Objective:                                                   -->
<!--                                                              -->
<!--    Caller requests FindHelp service for Lockout service      -->
<!--                                                              -->
<!--    This script                                               -->
<!--     sends a SIPINVITE to the FindHelp Service                -->
<!--     requesting that the group lockout_techies be used        -->
<!--     waits for “100” and “200” RC                             -->
<!--     issues a SIP ACK to acknowledge receipt of “200” RC      -->
<!--     waits for a SIP INFO with technician name and phone nr.  -->
<!--     displays popup using DOS MSG command                     -->
```

```
<!--     waits for BYE command that terminates the session        -->
<!--                                                              -->
<!--    The following strings must be changed before executing    -->
<!--    this script:                                              -->
<!--    <wtl_ipaddress> - ip-address of the test machine with the -->
<!--                    SIP softphone for Tech1 installed         -->
<!-- ´  <caller_phone_port> - port which the SIP softphone for    -->
<!--                    caller is listening                       -->
<!-- Date:                                                        -->
<!--    29-June-2006                                              -->
<!--                                                              -->
<!-- ############################################################ -->

 <scenario name="Caller requests FindHelp">
 <send retrans="500">
   <![CDATA[

      INVITE sip:FindHelp@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport]
[local_ip]:[local_port];branch=z9hG4bK4F58B3359B841
      From: caller
<sip:caller@<wtl_ipaddress>:<caller_phone_port>>;tag=[call_number]
      To: sut <sip:FindHelp@[remote_ip]:[remote_port]>
      Call-ID: [call_id]
      Cseq: 1 INVITE
      Contact: sip:caller@[local_ip]:[local_port]
      Max-Forwards: 70
      Subject: FindHelp
      P-Charging-Vector:
icid-value=294_1124116770286@47.135.114.87;orig-ioi=scscf1@homedomain.c
om
      Content-Type: text/xml
      Content-Length: [len]

     <?xml version="1.0" encoding="UTF-8"?>
        <FindHelp>
            <group>sip:lockout_techies@itso.ral.ibm.com</group>
        </FindHelp>


    ]]>
  </send>

  <recv response="100" optional="true" />
  <recv response="180" optional="true" />
```

```
    <recv response="200" rtd="true">
    </recv>

    <send>
     <![CDATA[

        ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0
        Via: SIP/2.0/[transport]
[local_ip]:[local_port];branch=z9hG4bK4F58B3359B842
        From: caller
<sip:caller@[local_ip]:[local_port]>;tag=[call_number]
        To: sut <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
        Call-ID: [call_id]
        Cseq: 1 ACK
        Contact: sip:caller@[local_ip]:[local_port]
        Max-Forwards: 70
        [$1]
        Subject: FindHelp
        Content-Length: 0



     ]]>
    </send>

    <recv request="INFO">
       <action>
            <ereg regexp="Ringing([[:alnum:]]*) ([[:alnum:]]*)
([[:alnum:]]*)" search_in="msg" assign_to


="3"/>
               <exec command="msg <windows_userid> Call in
progress:[$3]"/>
       </action>
    </recv>



     <send>
      <![CDATA[

        SIP/2.0 200 OK
        [last_Via:]
        [last_From:]
```

```
      [last_To:]
      [last_Call-ID:]
      [last_CSeq:]
      Content-Length: 0

    ]]>
   </send>
   <recv request="BYE"/>
   <send>
    <![CDATA[

      SIP/2.0 200 OK
      [last_Via:]
      [last_From:]
      [last_To:]
      [last_Call-ID:]
      [last_CSeq:]
      Content-Length: 0

    ]]>
   </send>


</scenario>
```

Note: The SIPp script for Caller_requests_FindHelp uses UDP/TCP port :5068 for SIP messages. The SIP softphone for the caller uses :5070.

It is important that the SIP port used by any particular component sending and listening for SIP messages is used only once for that physical machine. For example, if the SIP phone is using port :5070 on your Windows test server, then no other component may use :5070.

Start the WebSphere Process Server and the BPEL flow FindHelp

If you do not have WebSphere Process Server with the BPEL flow running, you must start it by performing the following:

1. Start WebSphere Integration Developer.
2. Open the **Business Integration** Perspective.
3. Click **Servers**.

4. Right-click **WebSphere Process Server V6.0**.
5. Select **Start**.

Publish the technician availability

Follow the steps for executing publish_technician1.xml and publish_technician2.xml which are detailed above in 13.3.2, “Use Case 2: Publish Technician Status” on page 469 to publish the technicians availability.

> **Tip:** The value set in Expires in the SIPp script limits the length of time that the technicians status remains active. In the example script Expires is set to 60 seconds. Therefore the scripts for publishing the status need to be executed immediately before executing the Caller_requests_FindHelp script.

Caller requests FindHelp for lockout service

Start the SIPp script for Caller_requests_FindHelp.xml.

1. Type the command in Example 13-8.
   a. Substitute <rb_scr_dir> with the directory where you have stored the modified Caller_requests_FindHelp.xml script.
   b. Substitute <lts_ipaddress> with the hostname or IP address of the machine which is hosting the SIP Servlet FindHelp. In this sample environment configuration this is the Linux test server.
   c. Substitute <fhss_port> with the TCP/UPD port number assigned to the FindHelp SIP Servlet for SIP messages on the SIP application server. The FindHelp SIP Servlet is listening on :5060 in this sample environment.

Example 13-8 Invoke the FindHelp caller requests script

```
sipp -p 5068 -sf "<rb_scr_path>\Caller_requests_FindHelp.xml"
-trace_msg -m 1 <lts_ipaddress>:<fhss_port>
```

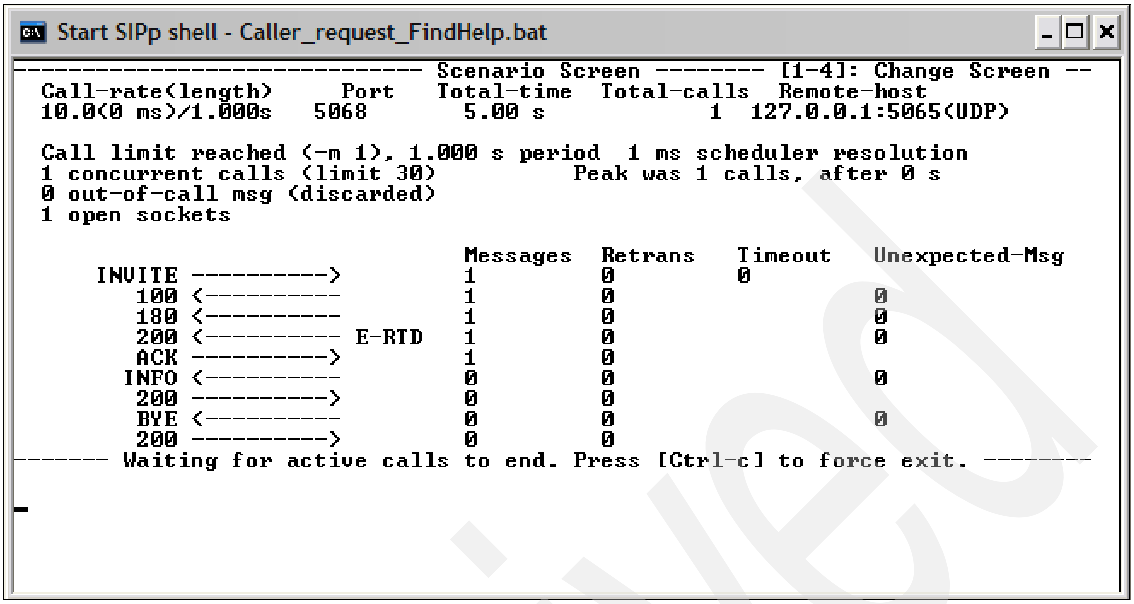


Figure 13-28 SIPp mainflow

Expected results

You may choose to watch the progress of the FindHelp BPEL flow in the WebSphere Process Server Console. Successful execution of FindHelp will produce a console log as shown in Figure 13-29.

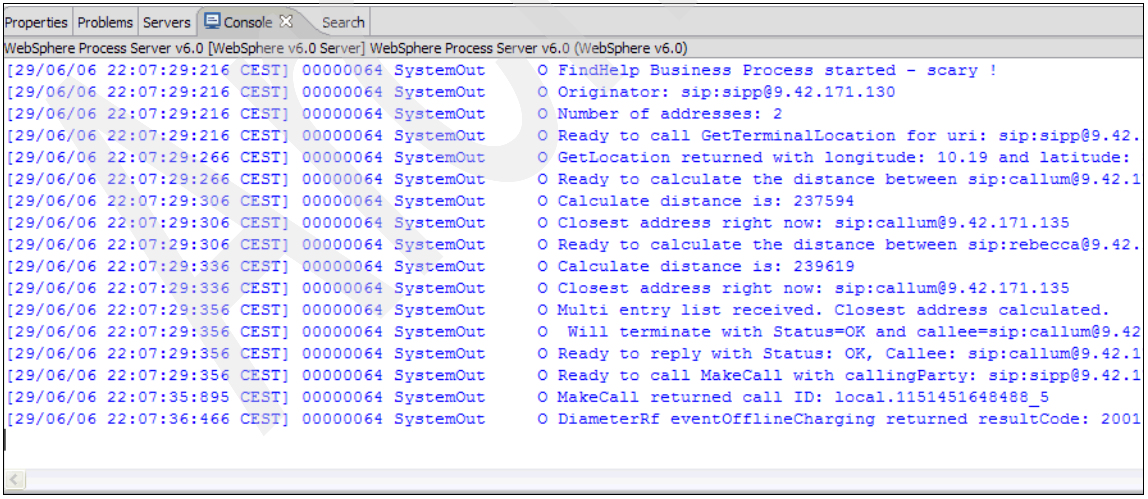


Figure 13-29 BPEL console output

After you receive the message `Ready to call MakeCall with callingParty` the two SIP softphones should ring. If you answer both phones, you will establish a call between the caller and the nearest active technician.

13.4 Problem determination and resolution

Should your results differ greatly from the expected use case result for Caller requests above, you may need to perform some problem determination to resolve any configuration, development or installation problems which exist in your setup.

For problem determination, it is useful to refer to the diagram in Figure 13-30 to follow the flow of the use case and trace it step-by-step.

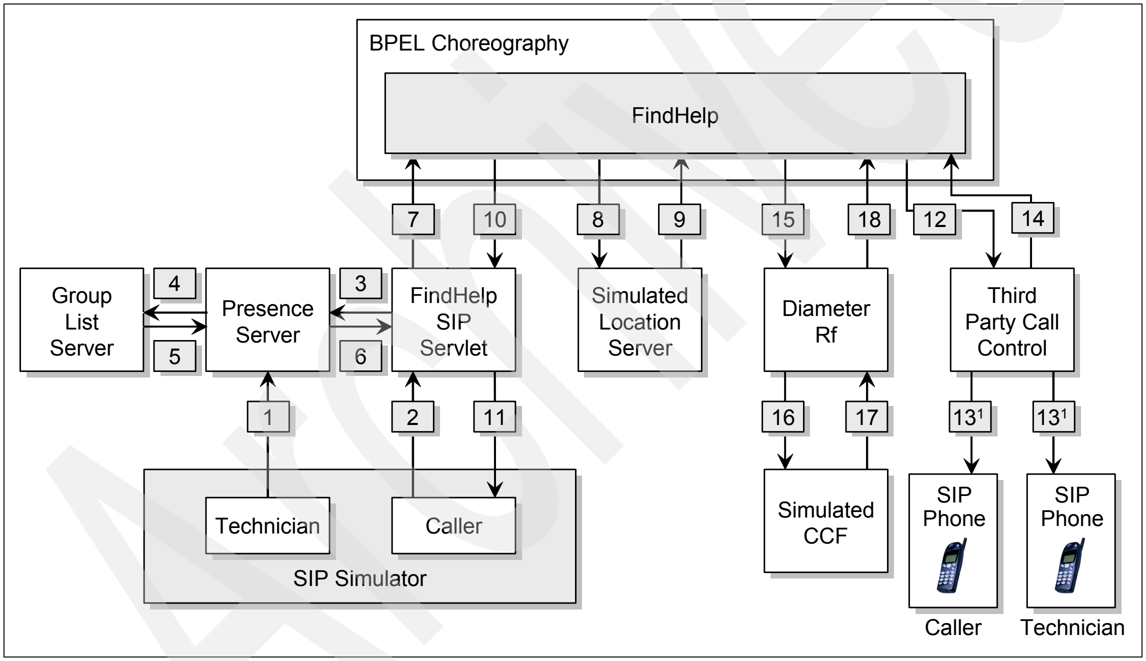


Figure 13-30 Caller requests FindHelp service Use Case Flow

13.5 Step-by-step tracing

Should you encounter problems, the following provides the desired log file messages for each step within the FindHelp service use case flow.

1. Technicians publish presence to Presence Server.

Check the SIPp log to verify that the outbound SIP **PUBLISH** is being sent and the corresponding **200 OK** is being returned.

If this is not the case, verify that the destination IP address and port of the FindHelp servlet is correctly configured.

Example 13-9 Desired message in SIPp log for Technicians publish presence

```
UDP message sent:

PUBLISH sip:service@9.42.170.173:5065 SIP/2.0
...
----------------------------------------------- 2006-06-22 10:43:17
UDP message received [299] bytes :

SIP/2.0 200 OK
...
```

2. FindHelp SIP Servlet is invoked by the SIPp script.

   Check the server log file for the application server node where the FindHelp SIP Servlet is installed. The file path for the Linux test server in this sample environment is `opt/IBM/WebSphere/AppServer/profiles/AppSrv01/`.

   If this step executes correctly your log file will contain the desired message as in Example 13-10.

Example 13-10 Desired message in the FindHelp SIP Servlet logfile

```
[29/06/06 16:28:55:366 EDT] 0000005f FindHelp      I   doInvite ENTRY()
[29/06/06 16:28:55:376 EDT] 0000005f FindHelp      I
contentBody='<?xml version="1.0" encoding="UTF-8"?>
```

3. FindHelp SIP Servlet subscribes to the Presence Server.

   Check the server log file for the application server node where the FindHelp SIP Servlet is installed. The file path for the Linux test server in this sample environment is `opt/IBM/WebSphere/AppServer/profiles/AppSrv01/`.

   If this step executes correctly your log file will contain the desired message as in Example 13-11.

Example 13-11 Desired message in the FindHelp SIP Servlet logfile

```
[29/06/06 16:28:55:836 EDT] 0000005f FindHelp      I generateSubscribe rc=true
```

4. The Presence Server subscribes to Lockout group to the Group List Server. This step allows for:

   – Retrieving group members
   – Subscription to group content changes

Group information is stored into the GLS.

Since both GLS and PS are installed on the same server it is not possible to check for exchanged SIP and HTTP messages using Ethereal.

Therefore we recommend enabling a detailed trace on both GLS and PS. To do so, enable the trace for GLS (`com.ibm.gml.*`) and/or for PS component (`com.ibm.workplace.* and com.ibm.wkplc.*`). Refer to 13.5.1, “Enable SIP debug tracing on the Linux test server” on page 486 for more information.

a. Click the select component, then **All Messages and Trace**.

b. Click **OK**, and then click **Save**.

5. FindHelp SIP Servlet has received response from the Presence Server.

   Check the server log file for the application server node where the Presence Server is installed. The file path for the Linux test server in this sample environment is `opt/IBM/WebSphere/AppServer/profiles/AppSrv03/`.

Example 13-12 Desired message in logfile for receiving response from Presence Server

```
[29/06/06 16:28:56:477 EDT] 00000064 FindHelp      I   doNotify() ENTRY
```

6. FindHelp SIP Servlet invokes BPEL.

   Check the server log file for the application server node where the FindHelp SIP Servlet is installed. The file path for the Linux test server in this sample environment is `opt/IBM/WebSphere/AppServer/profiles/AppSrv01/`.

Example 13-13 Desired message in FindHelp SIP logfile

```
[29/06/06 16:28:59:211 EDT] 00000064 FindHelp      I   invokeBPEL() ENTRY, 
originator=sip:sipp@9.42.171.130 availableGroupMembers=[sip:callum@9.42.171.135, 
sip:rebecca@9.42.171.135]
[29/06/06 16:28:59:241 EDT] 00000064 FindHelp      I   invokeBPEL() set URI for BPEL 
to=http://9.42.170.151:9081/FindHelpWeb/sca/setupcall
```

You can verify that the invocation request has been received by the WebSphere Process Server by checking the console for the messages indicated in the Figure 13-31 on page 482.

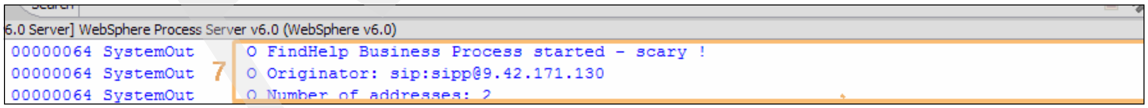


Figure 13-31 WebSphere Process Server console messages

If this is not the case, you should check the ProcessServerURI in the deployment descriptor for the FindHelp SIP Servlet to verify that the IP address for the WebSphere Process Server is set correctly.

7. FindHelp BPEL flow invokes the simulated Location Server.

   You can verify that the FindHelp BPEL flow invoked the simulated Location Server by checking the WebSphere Process Server console for the messages as listed in Figure 13-32.

```
00000064 SystemOut     O Ready to call GetTerminalLocation for uri: sip:sipp@9.42.171.130
00000064 SystemOut     O GetLocation returned with longitude: 10.19 and latitude: 89.0
00000064 SystemOut   8 O Ready to calculate the distance between sip:callum@9.42.171.135 and sip:sipp@9.42.171.
00000064 SystemOut     O Calculate distance is: 237594
00000064 SystemOut     O Closest address right now: sip:callum@9.42.171.135
00000064 SystemOut     O Ready to calculate the distance between sip:rebecca@9.42.171.135 and sip:sipp@9.42.171
```

Figure 13-32 Desired message in WebSphere Process Server console for FindHelp BPEL calling Location server

8. Simulated location server returns the results.

   When the simulated Location Server has successfully determined the location, similar messages to the ones in Figure 13-33 will appear in the WebSphere Process Server Console.

```
00000064 SystemOut     O Calculate distance is: 239619
00000064 SystemOut     O Closest address right now: sip:callum@9.42.171.135
00000064 SystemOut   9 O Multi entry list received. Closest address calculated.
00000064 SystemOut     O  Will terminate with Status=OK and callee=sip:callum@9.42.171.135
```

Figure 13-33 Desired message in WebSphere Process Server console on return from calling Location server

   If you do not see these messages, then you should investigate the following and ensure that:

   – The location server was installed and deployed on the Linux test server.

   – The end-point for the Location Server in the BPEL flow was set to the correct IP address of the location server.

   – The terminal_location.db file is in the /root directory on the Linux test server.

9. FindHelp BPEL Flow calls FindHelp SIP Servlet with member to be contacted.

   Verify that the FindHelp BPEL flow called the FindHelp SIP Servlet with the member to be contacted by checking the WebSphere Process Server console for the messages listed in Figure 13-34.

```
00000064 SystemOut     O  Will terminate with Status=OK and callee=sip:callum@9.42.171.135
00000064 SystemOut 10  O Ready to reply with Status: OK, Callee: sip:callum@9.42.171.135
00000064 SystemOut 12  O Ready to call MakeCall with callingParty: sip:sipp@9.42.171.130, calledParty: sip:cal
```

Figure 13-34 Desired message in WebSphere Process Server console

You can verify that the request from the FindHelp BPEL flow was received by the FindHelp SIP Servlet by checking the server log file for the application server node where the FindHelp SIP Servlet is installed. The file path for the Linux test server in this sample environment is: `opt/IBM/WebSphere/AppServer/profiles/AppSrv01/`

If this step executed correctly the log should appear as in Example 13-14.

Example 13-14 Desired messages in FindHelp SIP Servlet log upon return from FindHelp BPEL flow

```
[29/06/06 16:29:06:732 EDT] 00000064 FindHelp      I   invokeBPEL() Callee
returned='sip:callum@9.42.171.135 status=OK
[29/06/06 16:29:06:742 EDT] 00000064 FindHelp      I
memberToBeContacted=sip:callum@9.42.171.135 from the list of
availableMembers=[sip:callum@9.42.171.135, sip:rebecca@9.42.171.135]
```

If this is not the case, you should investigate the binding for the FindHelp SIP Servlet in the BPEL flow.

10. FindHelp SIP Servlet sends INFO to device client with who is going to contact them.

    Check the server log file for the application server node where the FindHelp SIP Servlet is installed. The file path for the Linux test server in this sample environment is `opt/IBM/WebSphere/AppServer/profiles/AppSrv01/`.

    If this step executed correctly the log should appear as in Example 13-15.

Example 13-15 Desired message in FindHelp SIP Servlet logfile

```
[29/06/06 16:29:06:742 EDT] 00000064 FindHelp      I   generateInfo ENTRY()
[29/06/06 16:29:06:762 EDT] 00000064 FindHelp      I   generateInfo EXIT() sent
```

11. Third Party Call Control calls Caller and Technician1.

    The Third Party Call Control now sends a SIP invite to the SIP softphones to establish a call between caller and Tech1. Both phones should ring. Should one or both phones not ring, you can verify whether the SIP INVITE arrived from the Third Party Call Control by capturing a SIP trace in Ethereal. Follow the instructions given in 13.5.2, “Tracing SIP messages using Ethereal” on page 488.

```
   21 24.421379   9.42.170.173        9.42.170.160        SIP/SD Request: INVITE sip:camer
   22 24.475063   9.42.170.160        9.42.170.173        SIP    Request: OPTIONS sip:9.42

⊞ Frame 21 (733 bytes on wire, 733 bytes captured)
⊞ Ethernet II, Src: Ibm_eb:32:a2 (00:0d:60:eb:32:a2), Dst: Ibm_48:01:88 (00:11:25:48:01:8
⊞ Internet Protocol, Src: 9.42.170.173 (9.42.170.173), Dst: 9.42.170.160 (9.42.170.160)
⊞ User Datagram Protocol, Src Port: 5069 (5069), Dst Port: 5060 (5060)
⊟ Session Initiation Protocol
  ⊞ Request-Line: INVITE sip:cameron@9.42.170.160:5060 SIP/2.0
  ⊟ Message Header
    ⊞ From: "sip:caller@9.42.170.160:5070" <sip:caller@9.42.170.160>;tag=4846557361317271_
    ⊞ To: "sip:cameron@9.42.170.160:5060" <sip:cameron@9.42.170.160>
      Call-ID: 32555528852269455@9.42.170.173
      Max-Forwards: 70
      CSeq: 2 INVITE
      Content-Type: application/sdp
      Content-Length: 194
      Via: SIP/2.0/UDP 9.42.170.173:5069;ibmsid=local.1149793899995_37_51;branch=z9hG4bK97
    ⊞ Contact: <sip:9.42.170.173:5069;transport=udp>
  ⊟ Message body
    ⊟ Session Description Protocol
        Session Description Protocol Version (v): 0
      ⊞ Owner/Creator, Session Id (o): sipX 5 5 IN IP4 9.42.170.160
        Session Name (s): phone-call
      ⊞ Connection Information (c): IN IP4 9.42.170.160
      ⊞ Time Description, active time (t): 0 0
      ⊞ Media Description, name and address (m): audio 8766 RTP/AVP 0 8 96
      ⊞ Media Attribute (a): rtpmap:0 pcmu/8000/1
      ⊞ Media Attribute (a): rtpmap:8 pcma/8000/1
      ⊞ Media Attribute (a): rtpmap:96 telephone-event/8000/1
```

Figure 13-35 Ethereal trace with SIP invite to initiate call between caller and tech1

If the SIP INVITE is in the Ethereal trace, and the phone did not ring, then check that both phones are listening on the ports which are set in their respective SIPp scripts.

a. Use the Windows Task Manager to determine the Process ID of the two SIP softphones.

b. Type the following command in a DOS command window to verify that the SIP phones are listening on the ports:

```
netstat -p UDP -ao
```

c. For the Process ID used by technician1's SIP softphone you should verify that the port for Technician1's SIP softphone is the same as `<tech1_phone_port>` in the SIPp script Publish_Technician1.xml.

d. For the Process ID used by the caller's SIP softphone you should verify that the port for caller is the same as `<caller_phone_port>` in the SIPp script Caller_requests_FindHelp.xml.

> **Tip:** We recommend that you configure the Linux test server and the Windows test machine on the same local area network, and not to traverse firewalls. Firewalls often do not allow UDP traffic and use Network Address Translation.

12. Diameter Rf calls CCF Simulator and returns.

    The Diameter Rf Web Service interface was called, creating a Charging Event Record in the CCF using the Diameter Protocol. The Diameter message should appear on the CCF Simulator display.

    If the call to the CCF Simulator failed to execute successfully, fault messages will appear on the WebSphere Process Server console similar to Figure 13-36.

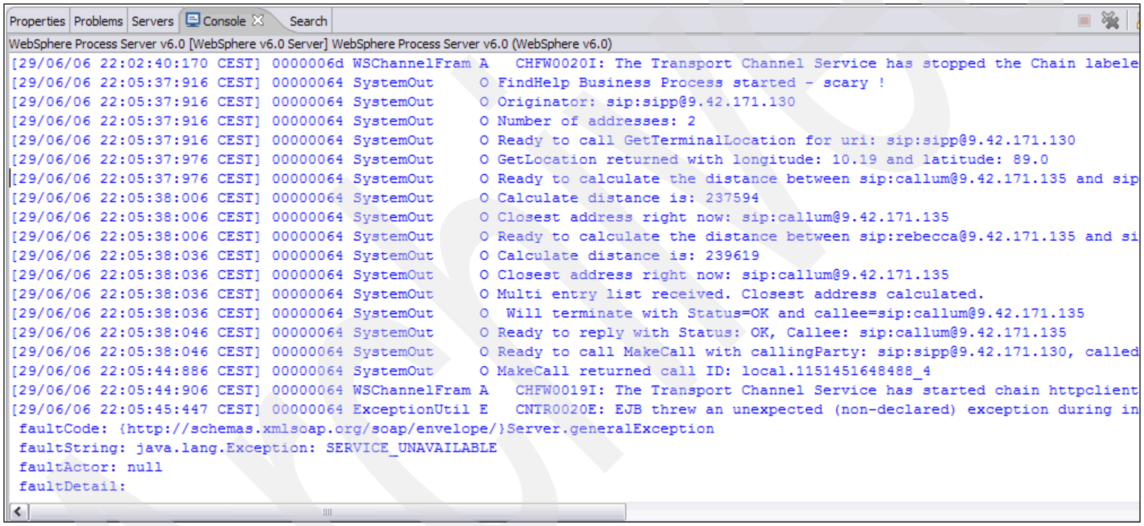


Figure 13-36 Messages in WebSphere Process Server Console if CCF not installed and configured correctly

13.5.1 Enable SIP debug tracing on the Linux test server

To trace on the Linux test server components, the SIP debug tracing mode must be enabled so trace messages can be written to the trace.log file for that application server node.

1. On the WebSphere Application Integration Solution Console for the Application Server Node select **Troubleshooting** → **Logs and Trace**.
2. Click **Server1**.

3. Click **Diagnostic Trace**.
4. Click **Change Log Detail Levels**.
5. Select the **Runtime** tab.
6. Enter the following in the **Groups** input window as in Figure 13-37 on page 487:

```
*=info:com.ibm.ws.sip.*=all: com.ibm.com.wsspi.sip.*=all:
com.ibm.ws.udp.*=all: com.ibm.wkplc.*=all: com.ibm.workplace.*=all
```

7. Click **OK**.

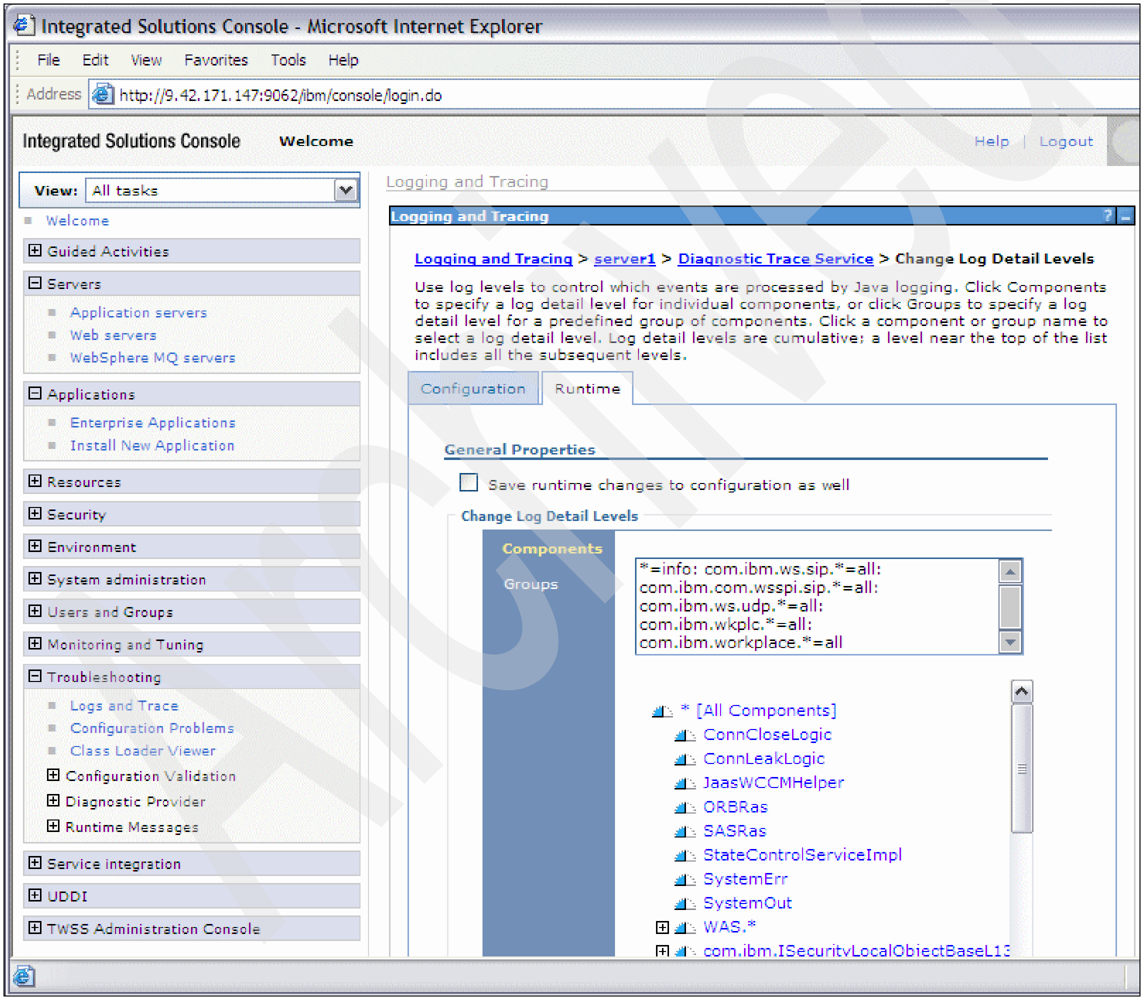


Figure 13-37 Enabling SIP debug tracing

13.5.2 Tracing SIP messages using Ethereal

You can enable SIP messaging tracing using Ethereal by performing the following steps:

1. Click **Start** → **Program** → **Ethereal** → **Ethereal** and you should see the main Ethereal screen as shown in Figure 13-38.

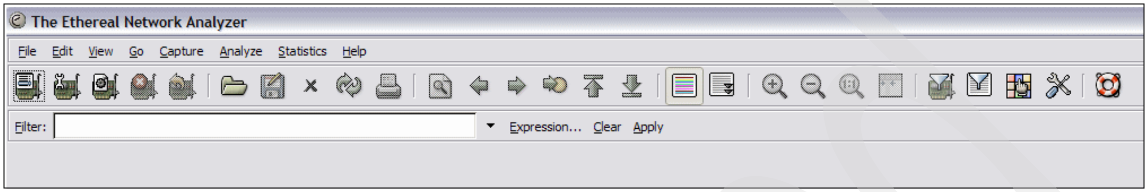


Figure 13-38 Starting Ethereal

2. Then click the **Capture** → **Interfaces** link at the top of your screen. You should get the popup menu as in Figure 13-39.

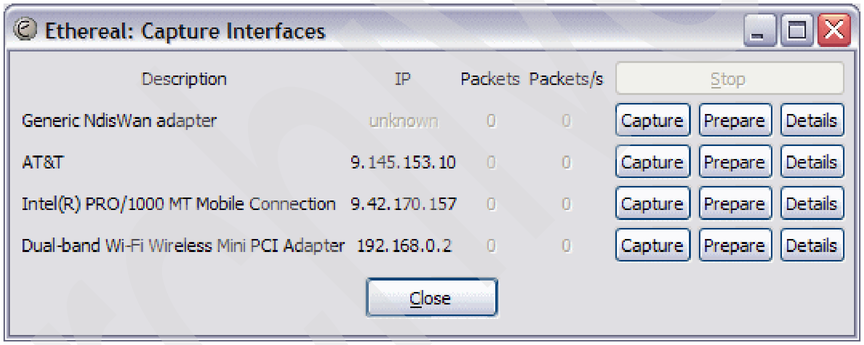


Figure 13-39 Capture Interfaces menu

3. Select the right network interface.
4. Click **Prepare**, and you should see the Capture Options menu as shown Figure 13-40 on page 489.

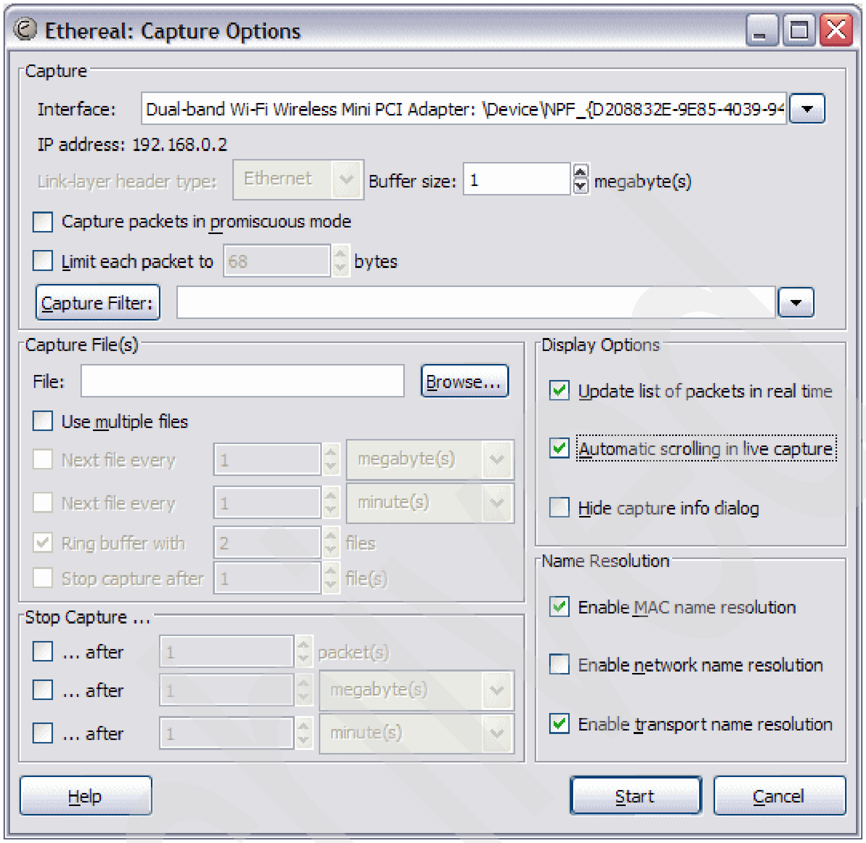


Figure 13-40 Capture Options menu

5. Deselect **Capture Packets in promiscuous mode**.
6. Select:
   a. **Update packets in real time**
   b. **Automatic scrolling in live capture**
7. Click **Start**.

   Ethereal should start capturing packets on the selected network interface and the main panel will be displayed as shown Figure 13-41 on page 490.

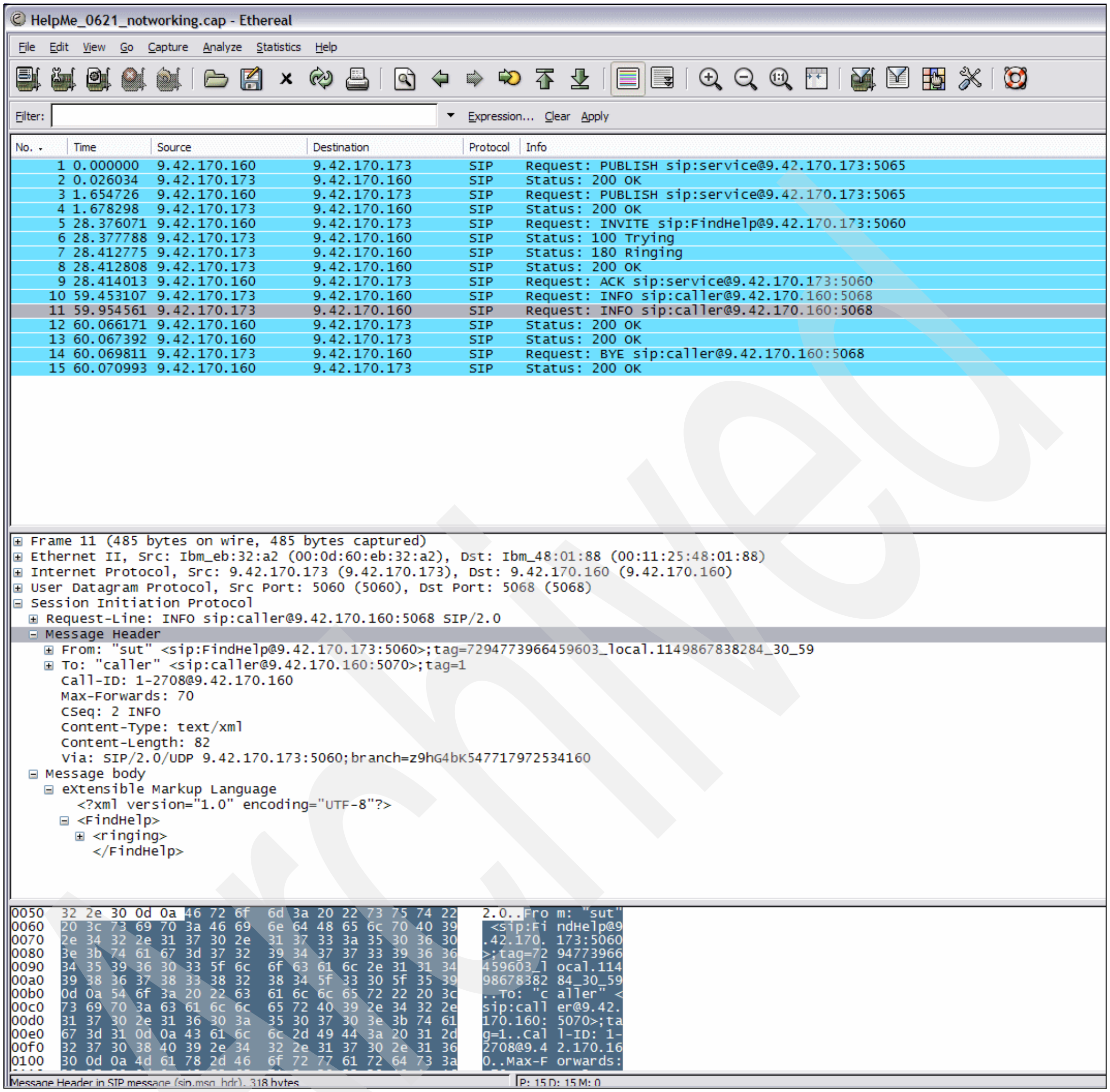


Figure 13-41 Capturing packets

A popup window showing percentage of captured packets as in Figure 13-42 on page 491 will also be displayed.

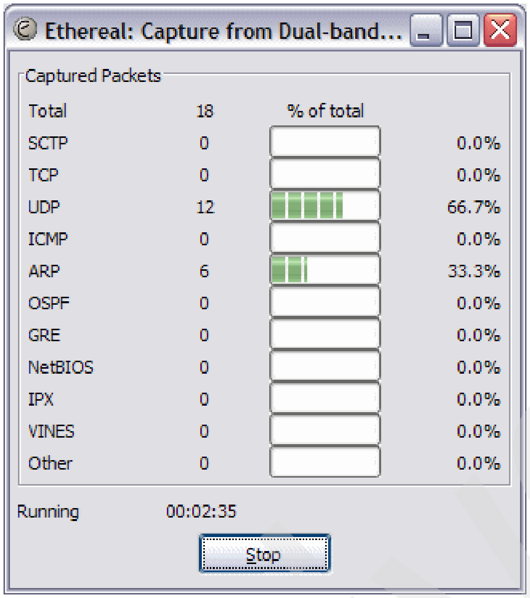


Figure 13-42 Statistics on captured packets

8. You are now looking at a status window that shows how many packets are being captured. By default Ethereal captures all packets coming from and going to your IP address. You might see a lot of traffic or just a little traffic depending upon how much is going on your network. If the reading is all zeros, try opening up a Web page. If you still do not see captured data, then you probably have the wrong Interface selected on the Capture options window. If you have too much traffic you can specify which protocol you're interested in by specifying the protocol name in the Filter field.

   a. Enter **sip**.

   b. Click **Apply**.

   Only SIP packets will be traced.

9. When you want to terminate the trace mode, click the **Stop** button on the Capture popup showing the statistics about captured packets.

10. Ethereal captures data that you can see on the main screen as shown Figure 13-41 on page 490. The screen contains three frames:

   – Frame 1

     Is at the top of the window. It shows an overview of the packets that were received or sent. The source column shows the IP address of the source of the packet. The destination column the IP address of the destination. The protocol column tells what protocol the packet was sent with. The info column lists the specific requests responses. It shows the SIP command, as well as the SIP URI contained in the SIP request or the response code in case of a SIP response. It also shows the ports the data traveled on.

 – Frame 2

 Is in the middle of the window. It shows more detailed information about a selected packet. You select a packet by clicking it.

 – Frame 3

 Is at the bottom of the window. It shows the data contained in the packet in hex format.

11. By clicking **Statistics** → **Flow Graph** → **General Flow** → **OK**, you can get a graphical display of the traced packets. Figure 13-43 on page 493 shows the output produced after tracing the sample “Find Help” application scenario on the device client side.

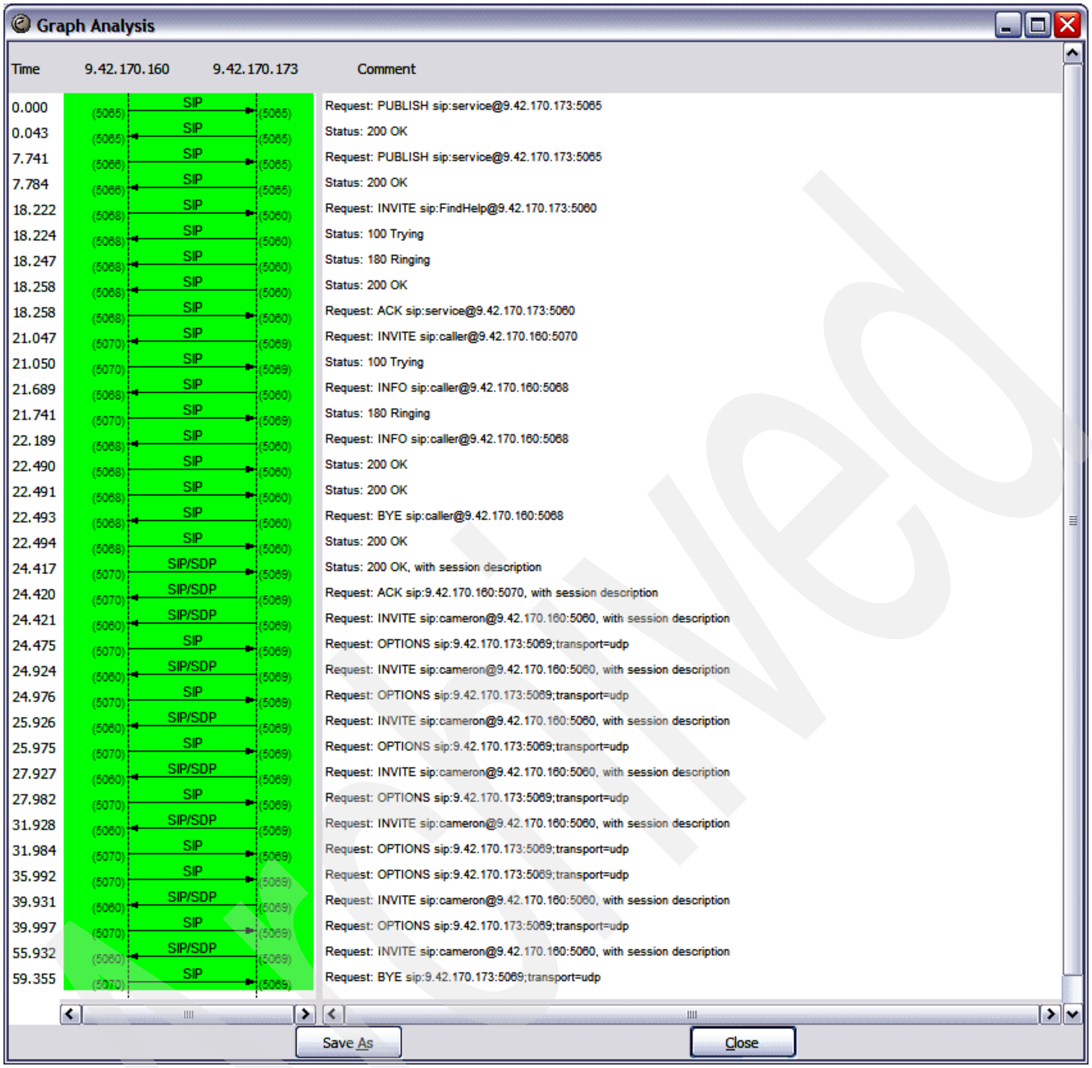


Figure 13-43 Graphical Analysis for the FindHelp scenario

13.6 Log files

A number of log files are maintained by the different components both on the Windows test client machine and on the Linux test server. The following are some of the log files that are available in the sample test environment.

WebSphere Process Server

WebSphere Integration Developer contains a server console for the WebSphere Process Server. Once the console display buffer is filled or a large number of messages have been issued, you need to view the logfile for messages. The log file is typically located in the following path:

```
<wid_root>pf\wps\logs\server1\startServer.log
```

The default directory, when installed on top of Rational Software Architect, is:

```
C:\Program
Files\IBM\Rational\SDP\6.0\pf\wps\logs\server1\startServer.log
```

SIPp Logs

SIPp logs are contained in the sub-directory where the SIPp XML scripts are stored. SIPp logs are written by default, so it is not necessary to enable logging.

sipXphone

The sipXphone logfiles are contained in the directory where sipXphone is installed.

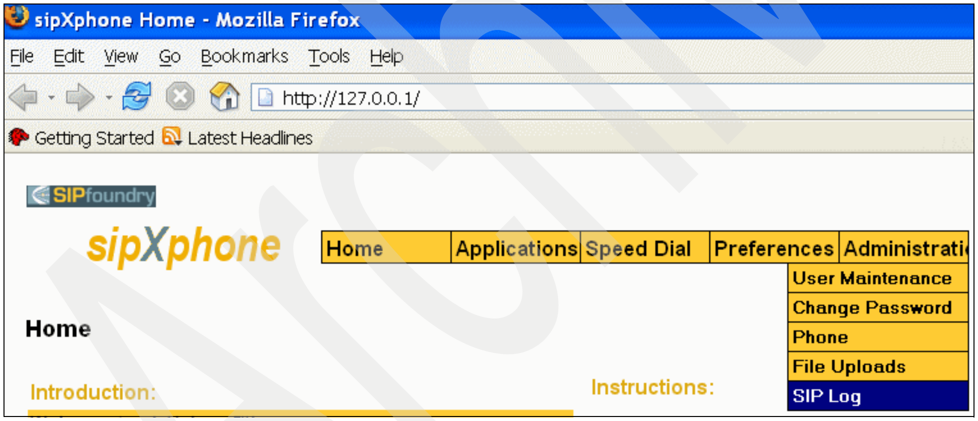


Figure 13-44 Access the sipXphone logging configuration

1. In the sipXphone Administration Web console:
   a. Select **Administration** → **SIP Log**.
   b. Click **Enable SIP Logging**.

   Logging for sipXphone is now enabled.

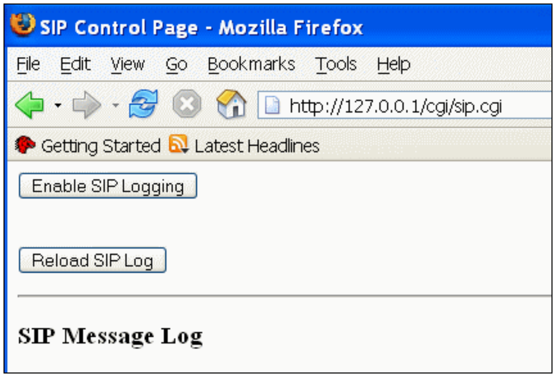


Figure 13-45 Enabling sipXphone Logging

After executing actions with the sipXphone, you can view the logs with the sipXphone Administration console.

To view the log, click **Reload SIP Log**.

SJPhone

To view the SJPhone logs:

1. Right-click the surface of the phone.
2. Select **To Advanced Mode**.
3. Right-click.
4. Select the option **Show Log**.

   The SJPhone log window should appear. When you make a call the log is displayed in this window.

5. If you want to clear or copy the log:

   a. Right-click within this window.

   b. Select **Clear** or **Copy**.

WebSphere Application Server

The System logs for the components running on the Linux test server are in separate logfiles for each application server profile. The logfile used depends upon which server node of the component that is installed.

The system logs are written to `<was_profile>/logs/server1/SystemOut.log`. Where, <was_profile> is the path for the application server profile. An example would be `opt/IBM/WebSphere/AppServer/profiles/AppSrv02`.

The following table provides the system log paths configured for this sample environment setup. The paths in your environment setup may be different due to your WebSphere Application Server configuration and the components you installed on each application server.

Table 13-2 Linux test server logfiles

Runtime components		**Logfile path**
WebSphere Application Server 6.1 AS 1	FindHelp SIP Servlet Simulated Location Server	opt/IBM/WebSphere/AppServer/profiles /AppSrv01/
WebSphere Application Server 6.1 AS 2	Group List Server	opt/IBM/WebSphere/AppServer/profiles /AppSrv02/
WebSphere Application Server 6.1 AS 3	Presence Server	opt/IBM/WebSphere/AppServer/profiles /AppSrv03/
WebSphere Application Server 6.1 AS 4	Diameter RF	opt/IBM/WebSphere/AppServer/profiles /AppSrv04/
WebSphere Application Server 6.1 AS 5	3rd Party Call Control	opt/IBM/WebSphere/AppServer/profiles /AppSrv05/

Part 5

Appendixes

Part 5 provides additional information about the installation and configuration of the test environment and references to related publications and other useful information, including how to obtain the source code for the sample applications.

A

Installing the application development environment

This appendix describes the process for installing the different tools that were used in this redbook to develop the SIP and IMS sample applications.

This appendix contains the following:

- Installing the SIP AST
- SIP device client installation
- Installing the IMS Enablement Toolkit
- Installing WebSphere Integration Developer
- Installing the Telecom Web Services Server plug-in

A.1 Installing the SIP AST

This section provides instructions on how to install the WebSphere Application Server Toolkit (AST). AST is packaged with the WebSphere Application Server V6.1, so you initiate the installation through the WebSphere Application Server LaunchPad.

1. Double-click the Launchpad.exe.
2. Select Application Server Toolkit Installation from the options on the left side of the window.

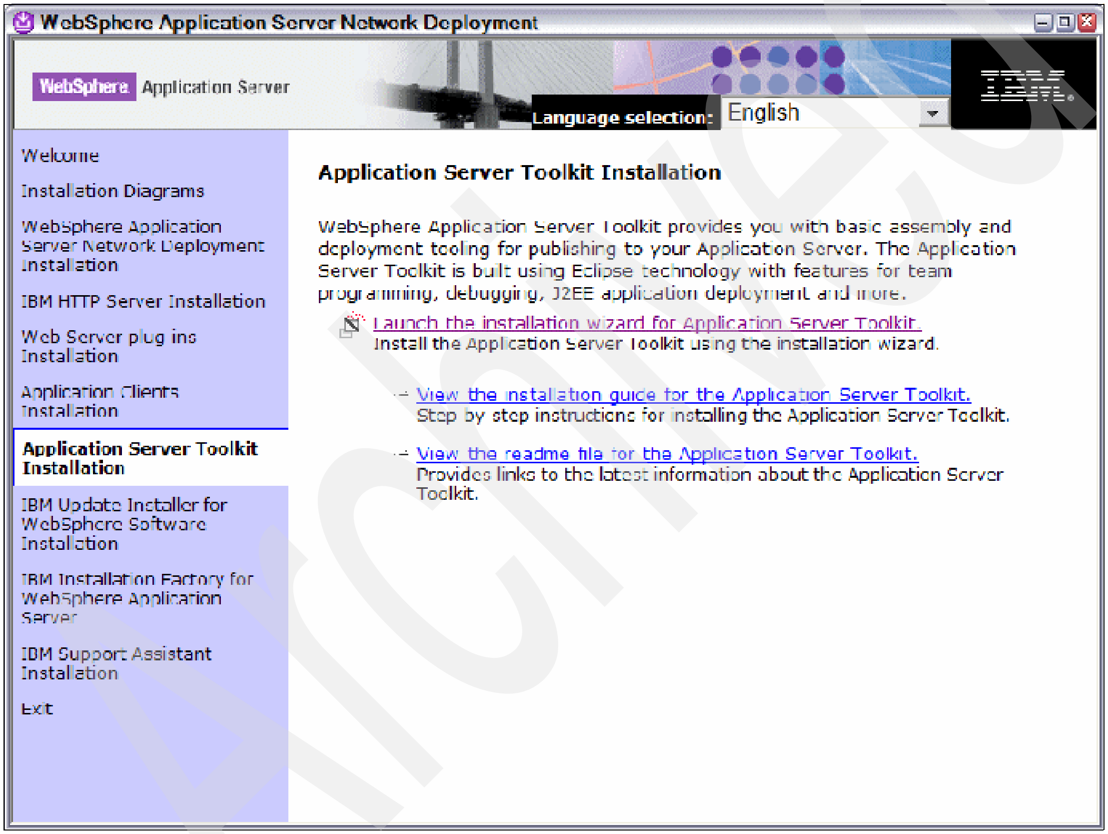


Figure A-1 Launchpad for WebSphere Application Server Network Deployment

3. Click **Launch the installation wizard for Application Server Toolkit** link.

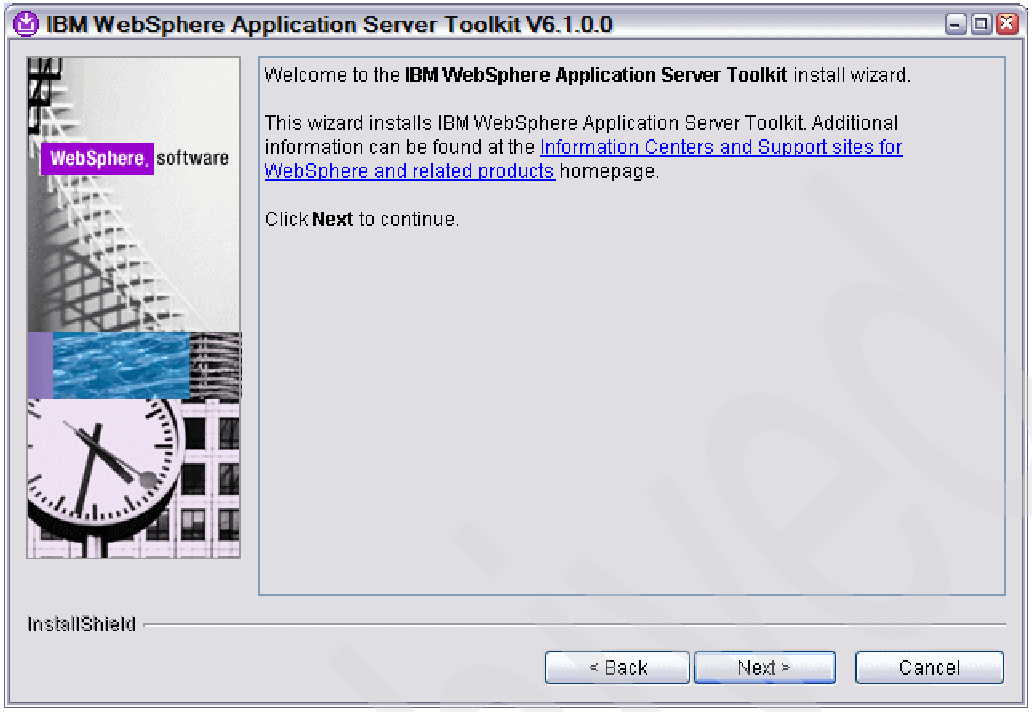


Figure A-2 IBM WebSphere Application Server Toolkit install wizard

4. Click **Next**.
5. Accept the License Agreement and click **Next**.

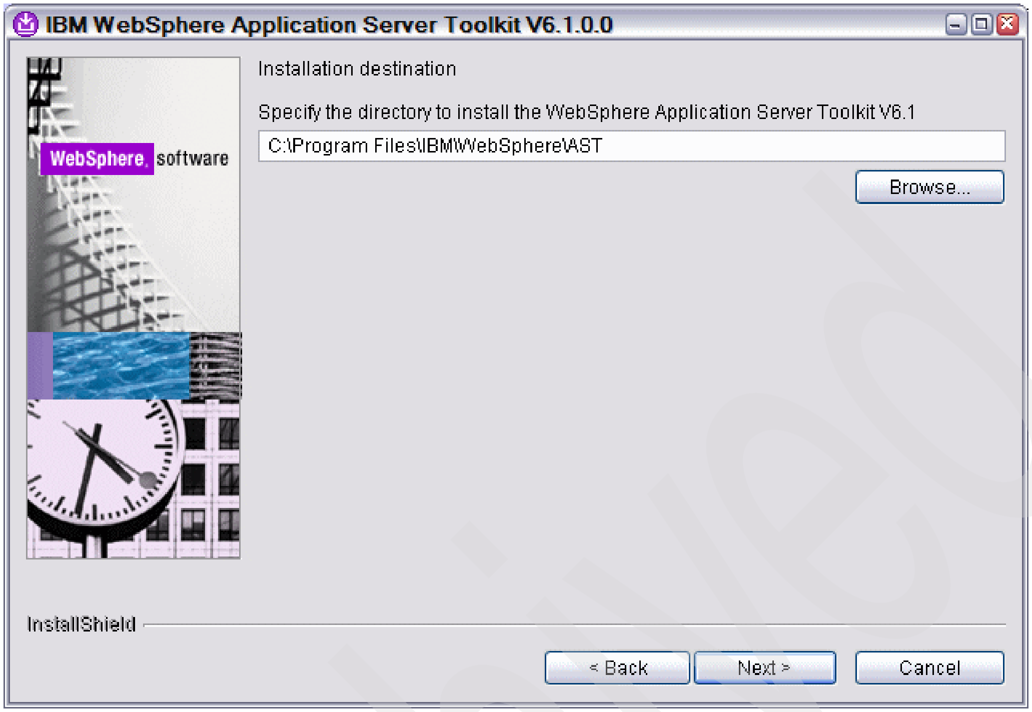


Figure A-3 Specifying the installation destination

6. You can change the installation destination directory if you so desire.
7. Click **Next**.
8. Review the summary information.
9. Click **Next** to begin the installation.
10. Once this has completed, click **Finish**.

A.1.1 Starting the SIP AST

Start the SIP AST by selecting **Start** → **All Programs** → **IBM WebSphere** → **Application Server Toolkit V6.1** → **Application Server Toolkit**

A.2 SIP device client installation

Three softphones were used to simulate call-setup and voice traffic for the SIP sample applications development and the IMS sample application scenarios. This section describes the installation process for each of the softphones.

Note: The configuration of the each softphone is explained in the respective chapters where each phone is used.

A.2.1 SipXphone

The sipXphone is used for both the SIP and the IMS sample application development scenarios. This section provides instructions on the basic installation of sipXphone.

1. The file sipXphone-2_6_0_27.exe is available as part of the additional materials that you can download for this redbook (see Appendix C, "Additional material" on page 637).

   Note: sipXphone version 2.6 for Windows was installed for this redbook.

2. Install SipXPhone to the Windows test machine.

   a. Copy the SipXPhone install file sipXphone-2_6_0_27.exe to a temporary directory on the Windows test machine.

   b. Double-click the file sipXphone-2_6_0_27.exe.

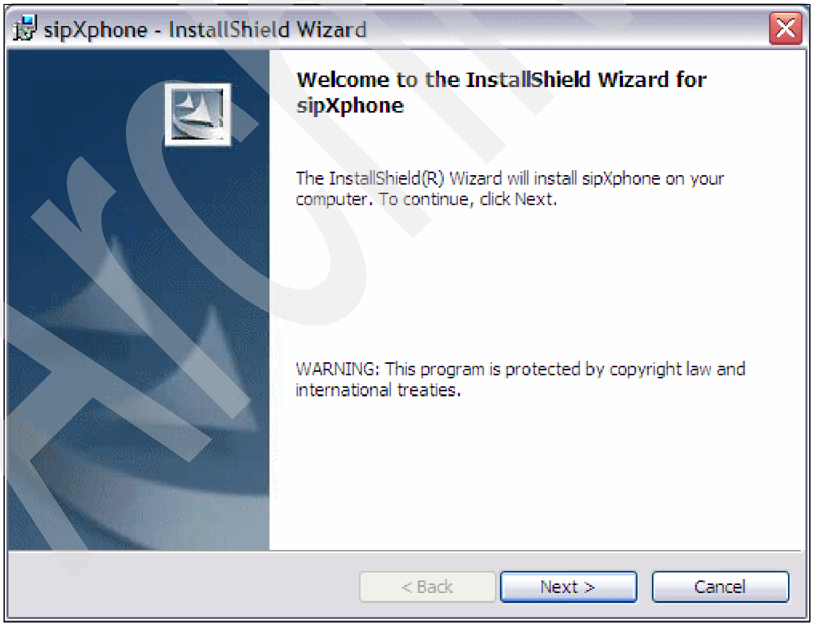


Figure A-4 Welcome to the InstallShield Wizard for sipXphone

   c. Click **Next**.

d. Accept the license agreement and click **Next**.

e. Accept the default destination location and click **Next**.

f. Review the installation settings .

g. Click **Install**.

h. When the **InstallShield Wizard Completed** window appears, click **Finish**.

3. Verify the sipXphone installation. From the Windows **Start** → **Programs** menu, select **SIPfoundry** → **sipXphone** → **sipXphone**.

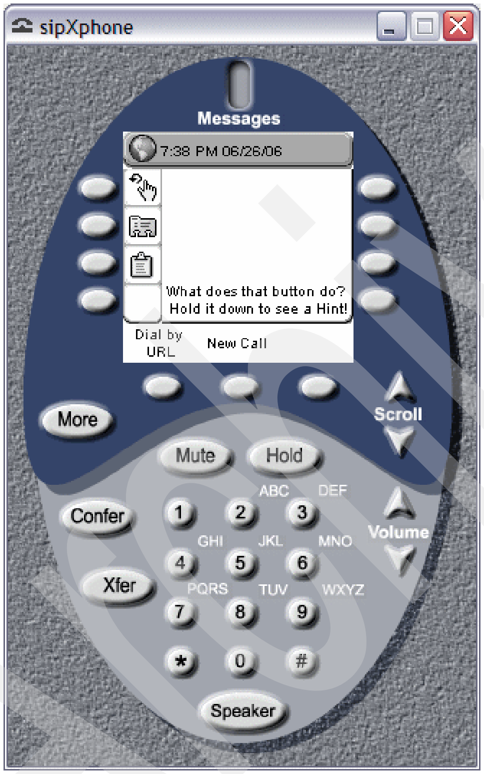


Figure A-5 Verify sipXphone installation

4. Enable Administration Web console.

a. Click **More** → **Prefs**.

b. Click **mysip softphone Web**.

c. Leave the admin password blank.

d. Click **OK** in the **Authentication** window.

e. Click the box to the right of **Enable Web Server?**

f. Accept 80 as the **HTTP Port**.

g. Click **OK**.

5. Click **Restart** to restart the sipXphone.

A.2.2 X-Lite

X-Lite is the second SIP softphone which was used for the SIP development scenarios. This section provides instructions on the installation of X-Lite.

1. Download X-Lite from:

   http://www.counterpath.com/index.php?menu=download

   Note: X-Lite version 3.0 for Windows was installed for this redbook.

2. Install X-Lite to the Windows test machine.

   a. Copy the X-Lite install file X-Lite_Win32_1002tx_29712.exe to a temporary directory on the Windows test machine.

   b. Double-click the filename X-Lite_Win32_1002tx_29712.exe.

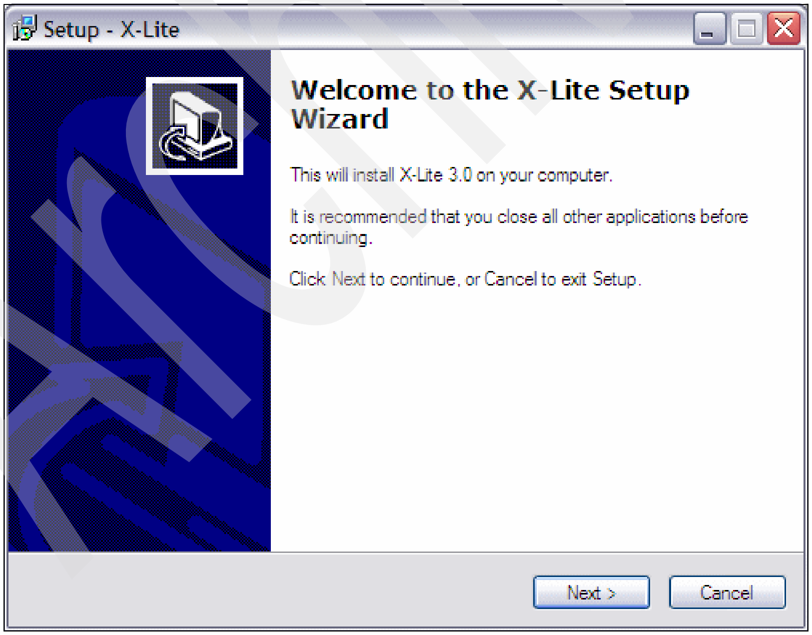


Figure A-6 Welcome to the X-Lite Setup Wizard

c. Click **Next**.

d. Accept the license agreement and click **Next**.

e. Accept the default destination location and click **Next**.

f. Accept the defaults on the **Select Additional Tasks** window and click **Next**.

3. When the installation is complete, if a window appears asking for restarting your computer:

   a. Select **No, I will restart the computer later**.

   b. Click **Finish**.

4. Verify the installation of X-Lite.

   a. Select **X-Lite** → **X-Lite** from the Windows **start** → **Programs** menu.

   b. Click **Close** on the **SIP Accounts** window.

   c. The X-Lite phone will appear with the message **No SIP accounts are enabled**.

Figure A-7 X-Lite Installation Verification

   d. Close the X-Lite phone by clicking **X** at the top of the phone.

   e. Click **OK**.

f. if a **Confirm to Quit** window appears, check the box **Do not show this dialog again** so the Confirm to Quit dialog won't appear again.

A.2.3 SJPhone

This section provides instructions on the installation of SJPhone. It is the third SIP softphone which was used for the SIP and IMS sample applications development.

1. Download SJPhone from:

   http://www.sjlabs.com/

   Note: SJPhone for Windows v.1.60.289a, 06.19.05 was installed for this redbook.

2. Install SJPhone to the Windows test machine.

   a. Copy the SJPhone install file **SJphone-289a.exe** to a temporary directory on the Windows test machine.

   b. Double-click the filename **SJphone-289a.exe**.

Figure A-8 Welcome to the SJPhone Installation Wizard

   c. Click **Next**.

d. Accept the license agreement and click **Next**.

e. When SJphone is finished installing, click **Finish.** This will start the SJphone.

3. Setup the installed SJphone.

   a. When the Audio Wizard window appears, click **Next**.

   b. Accept the default **Use DirectX®**, click **Next**.

   c. Accept the defaults for audio devices, click **Next**.

   d. Adjust playback volume, if desired, and click **Next**.

   e. Adjust the microphone recording level, if desired, and click **Next**.

   f. Verify the settings and click **Next**.

   g. Click **Finish**.

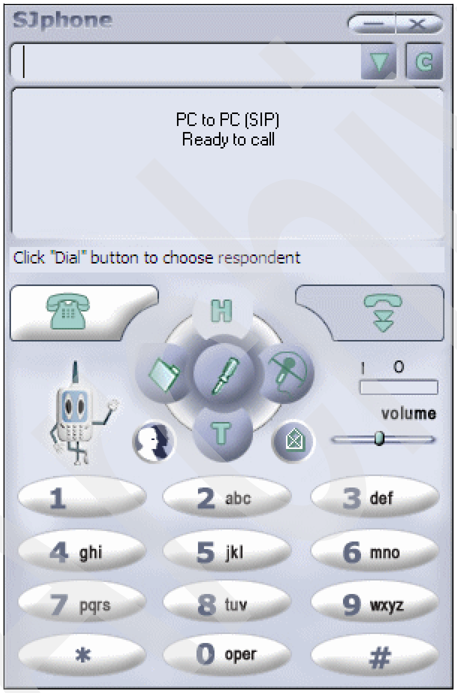


Figure A-9 SJPhone is installed

A.3 Installing the IMS Enablement Toolkit

Before installing the IMS Enablement Toolkit, you must have successfully installed and tested AST V6.1 (which includes the SIP Toolkit). If AST is not launched restart it as described above in A.1.1, “Starting the SIP AST” on page 502

You are now ready to install the IMS Enablement Toolkit. Start by locating the directory where you have saved the file **ims_wtp_update.zip**

1. From the AST top menu bar click **Help** → **Software Updates** → **Find and Install**.

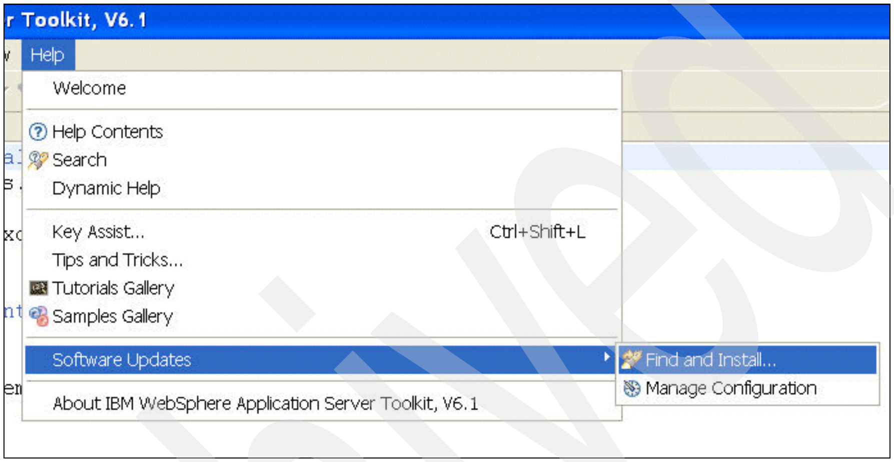


Figure A-10 Software Updates Find and Install

2. In the **Feature Update** window select **Search for new features to install**.
3. Click **Next**.
4. In the **Update Sites to Visit** window:
   a. Click **New Archived Site**.
   b. Navigate to the folder where you have saved **ims_wtp_install.zip**.
   c. Click **Open**.
5. In the Edit Local Site pop-up window, click **OK**.

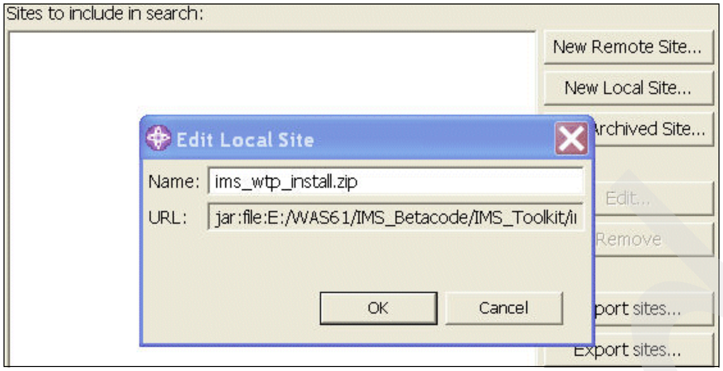


Figure A-11 Edit Local Site pop-up

6. Verify that **ims_wtp_install.zip** is selected and click **Finish**.
7. In the **Search Results** window.
   a. Select **ims_wtp_install.zip**.
   b. Verify that **Show the latest version of a feature only** is selected.
   c. Click **Next**.

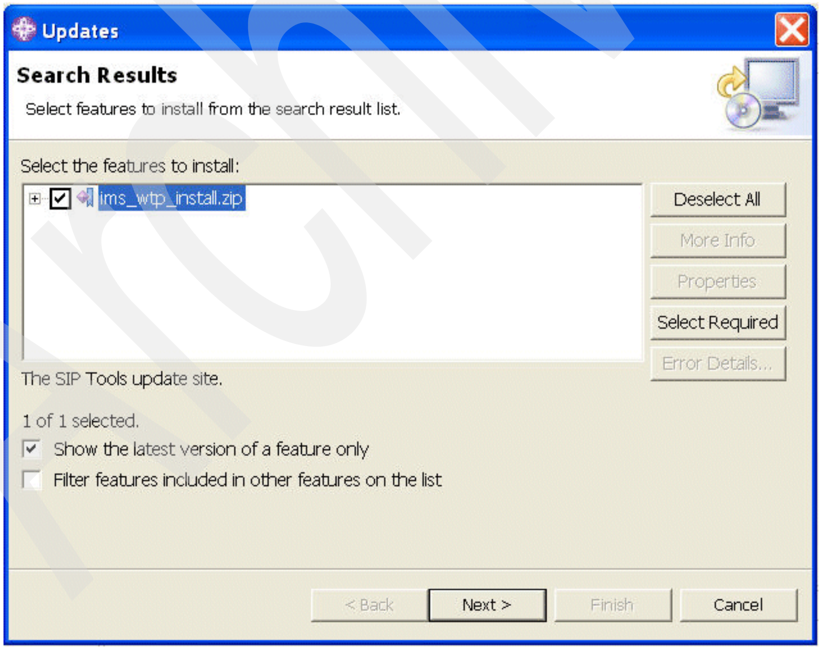


Figure A-12 Features Search Results

8. Accept the terms in the feature license agreement and click **Next**.

9. After verifying the installation location of the feature **IMS WTP Tools**, click **Finish**.

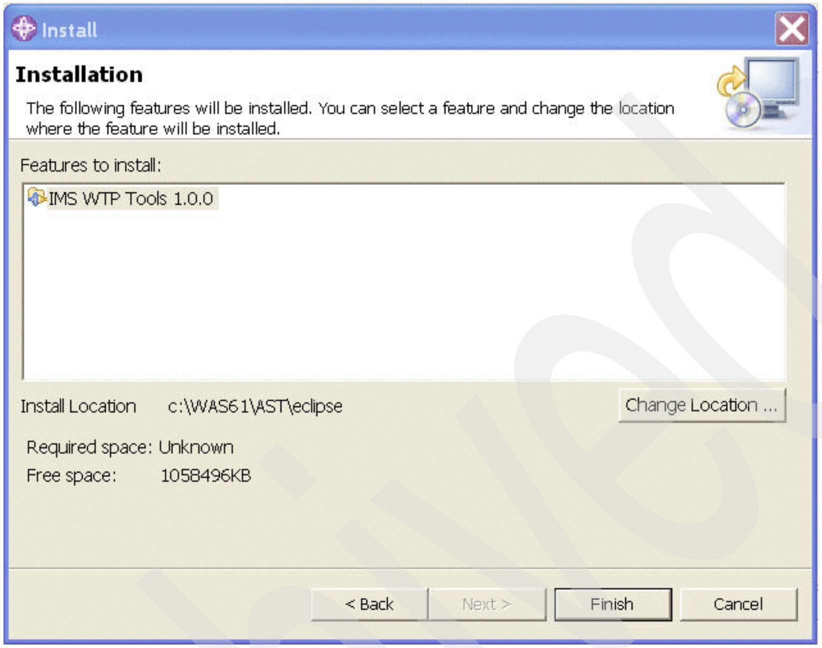


Figure A-13 Verify Install Feature Updates

10. Accept the feature license agreement and click **Next**.

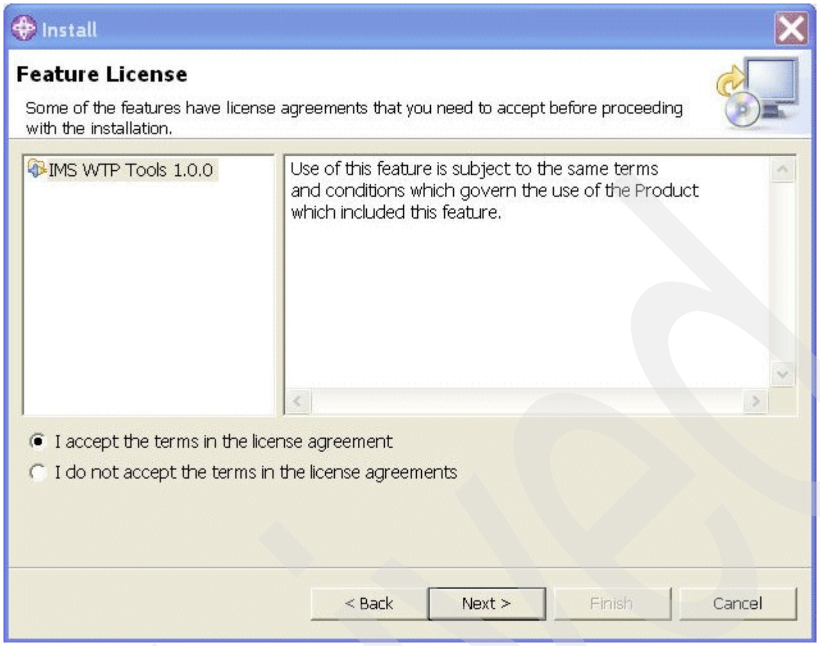


Figure A-14 Feature Licence Agreement

11. Click **Finish** to confirm the installation.
12. In the Installation confirmation window, click **Install**.
13. In the feature verification window, click **Install** to install **com.ibm.imstools_1.0.0**.

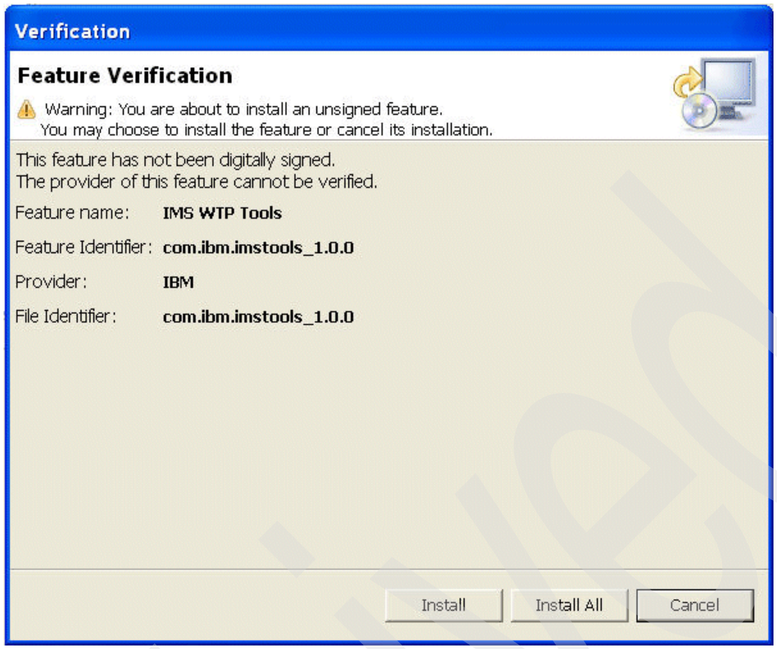


Figure A-15 Verify unsigned feature

14. At the end of the install process, click **Yes** to accept the restart of the workbench.

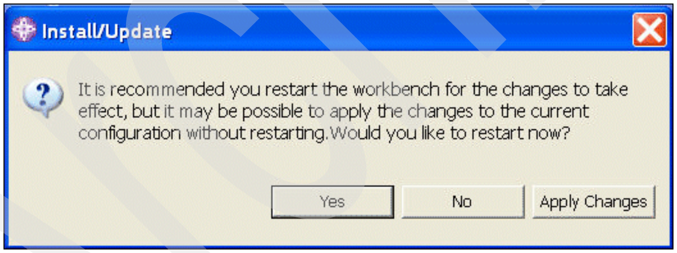


Figure A-16 Completed installation window

A.3.1 Verify the installation of the IMS Enablement Toolkit

1. From the AST top menu bar, click the **Help** → **Software Updates** → **Manage Configuration** menu.

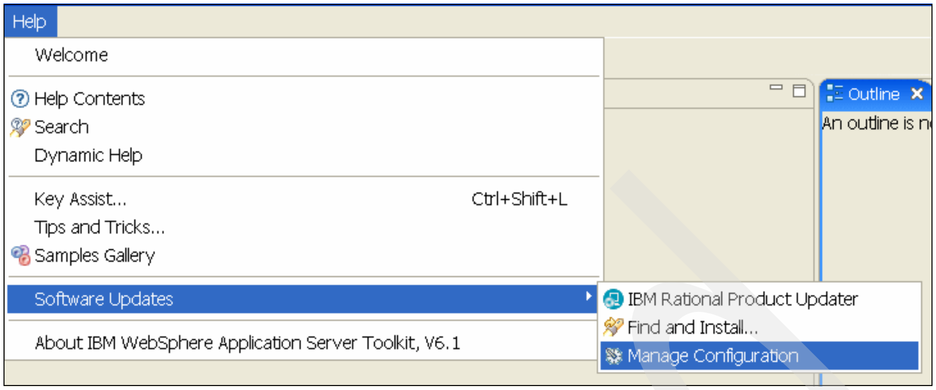


Figure A-17 Manage AST Configuration

2. Expand the **eclipse** folder (the eclipse name will be prefixed by the installation folder name of the place you installed the AST).
3. Select **IMS WTP Tools 1.0.0**.
4. In the right window, click **Show Properties**.
5. It should open a new window where you can select **General Information**.
6. You should see **Properties for IMS WTP Tools** similar to Figure A-18 on page 515 to confirm that the IMS Toolkit is correctly installed.

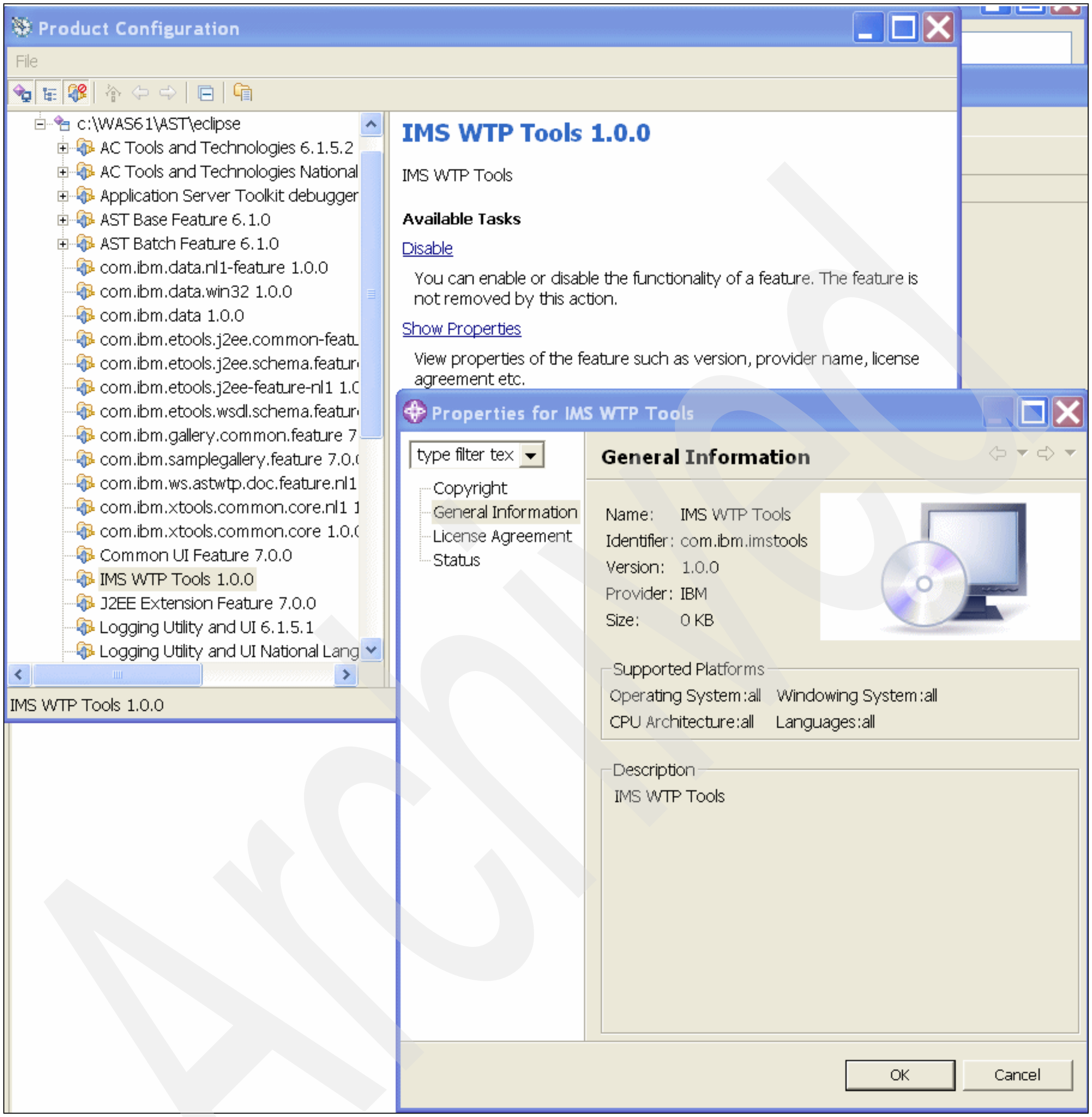


Figure A-18 Properties for IMS WTP Tools

A.4 Installing WebSphere Integration Developer

You start the installation by running the WebSphere Integration Developer **launchpad.exe**.

Note: If you use the `extractor.exe` to create an installation image the launchpad will load automatically once the extraction of the images is finished.

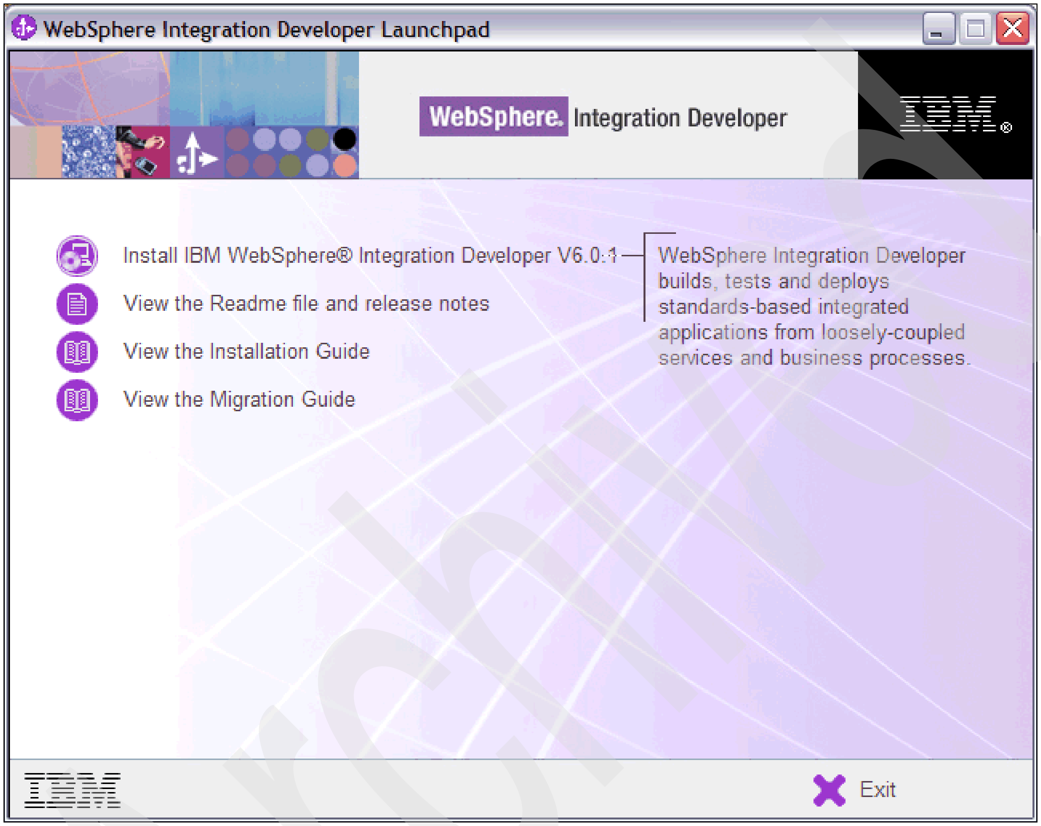


Figure A-19 The WebSphere Integration Developer Launchpad

1. From the launchpad, select **Install IBM WebSphere Integration Developer v6.0.1**.
2. The installer welcome screen will appear. Click **Next** to start the installation.
3. Accept the license agreement. Click **Next**.
4. Choose the appropriate installation directory. Click **Next**.

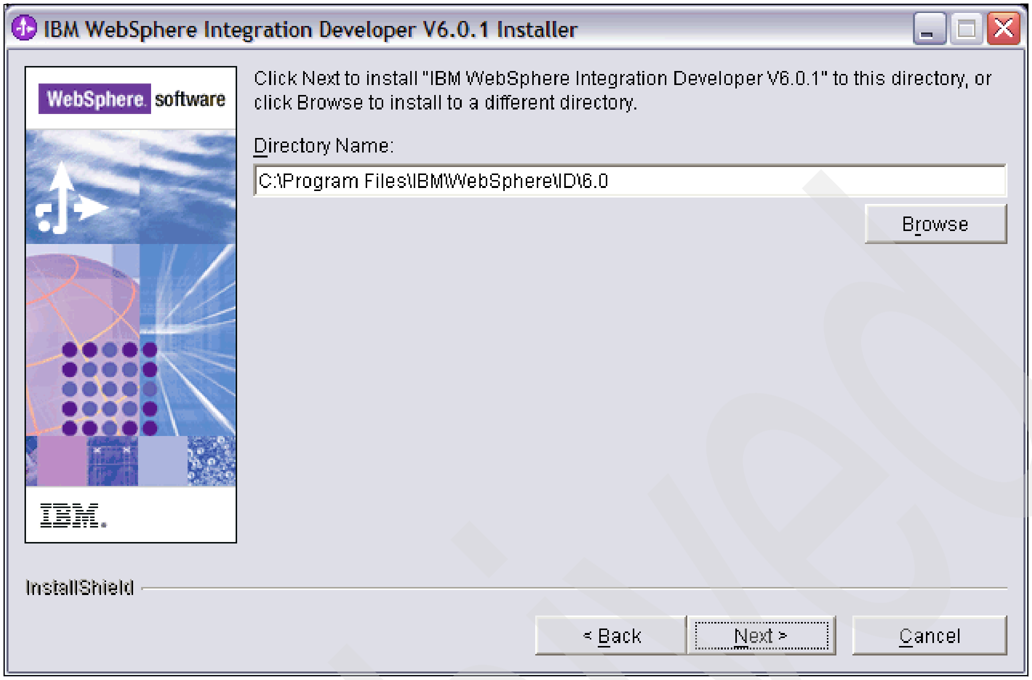


Figure A-20 Select the installation directory

> **Note:** A dedicated stand-alone test environment is not covered in this redbook. If you have already deployed a WebSphere Process Server and want to use it to test your application, then skip Step 5. Refer to the instructions about how to deploy your applications to this test environment in Chapter 16 of the redbook *Getting Started with WebSphere Integration Developer and WebSphere Process Server*, SG24-7130.

5. Check the **Integrated Test Environment** check box.
6. Click **Next**.

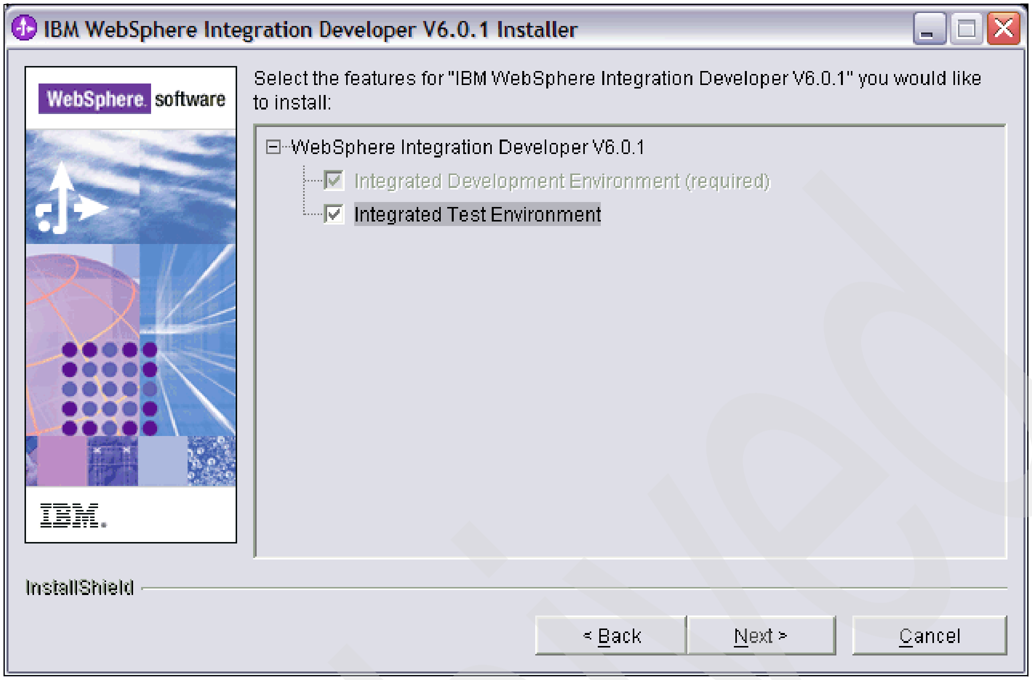


Figure A-21 Select the environments

7. Select the test environment profiles.
   a. Ensure that both check boxes **WebSphere Process Server** and **WebSphere Enterprise Service Bus** are checked.
   b. Click **Next**.

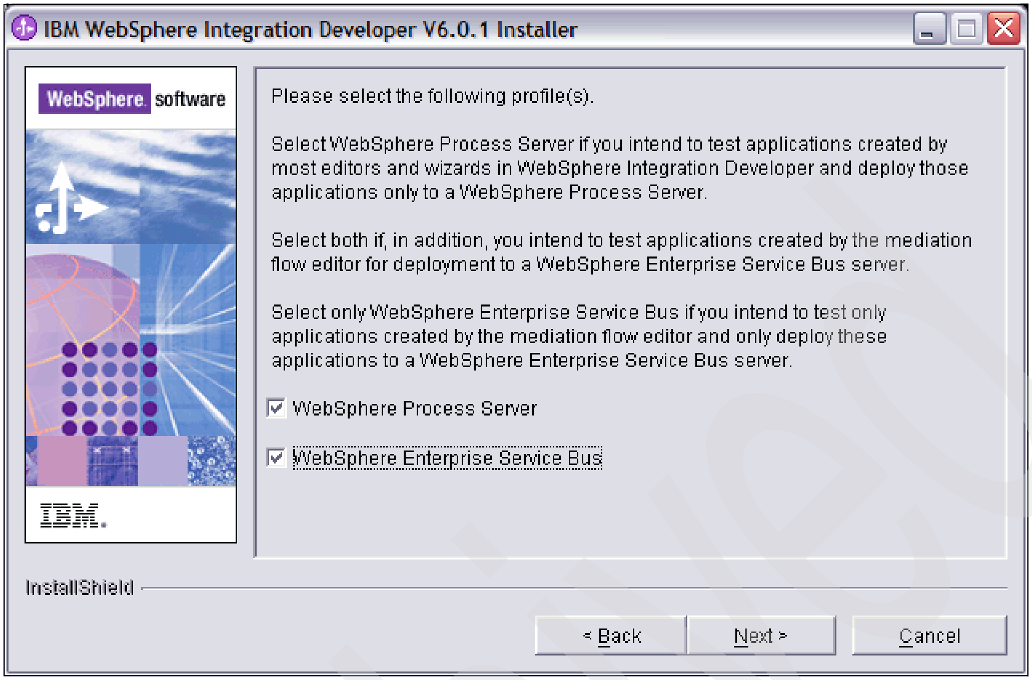


Figure A-22 Select the test environment profiles

8. A summary of your selected installation options displays. Click **Next** to start the installation.
9. When the installation is completed a window will appear summing up the installation results. Click **Next**.

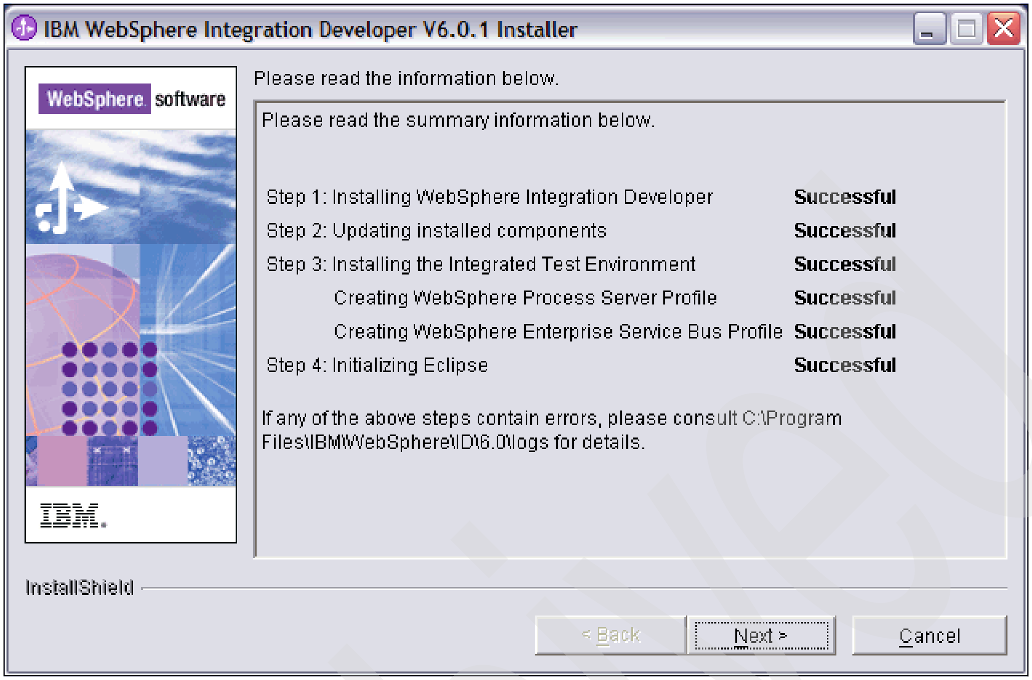


Figure A-23 Installation Summary

10. If you are connected to the Internet, select the **Launch Rational Product Updater** check box.
11. Click **Finish**.

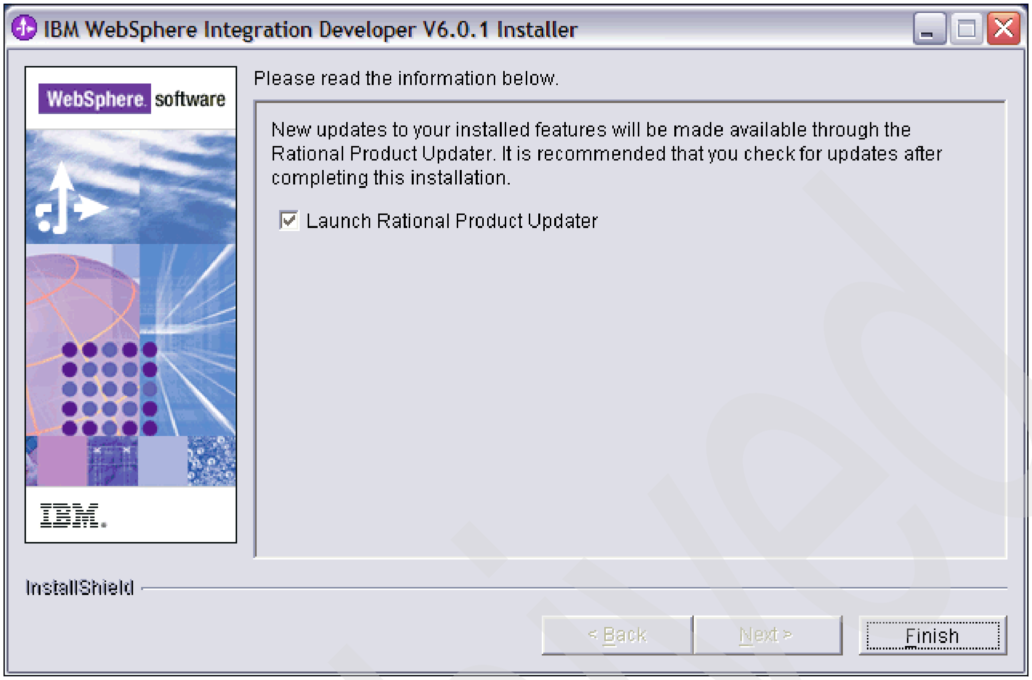


Figure A-24 Launch the product updater

A.4.1 Update WebSphere Integration Developer

The product update process will be launched on completion of the installation process if you are connected to the Internet and checked the **Launch Rational Product Updater** check box.

> **Note:** If you do not have access to the Internet, you can run the product update at a later point by selecting **Help** → **Software Updates** → **IBM Rational Product Updater**.
>
> If you prefer to download the fixpack 6.0.1.1 for later update, go to `http://www.ibm.com/software/integration/wid/support` and select the **Recommended Fixes** link, then click **v6.0.1.1**. At the bottom of the page you can download the “Installation Instructions” and **wid_6011_fixpack.zip**.

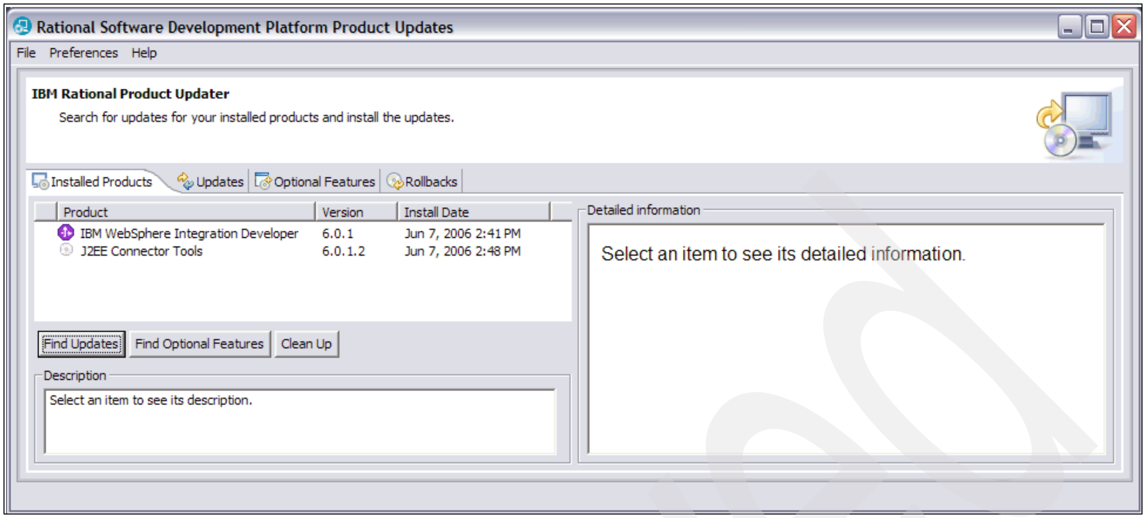


Figure A-25 The IBM Rational Product Updater

The Installed **Products Tab** lists all currently installed products. To search for available updates for these products click **Find Updates**. The product updater will connect to the IBM update site and retrieve all available product updates. These are listed in the **Updates tab**.

Select the relevant updates and click Install Updates. This starts the product update.

A.4.2 Apply required fixes

Follow these instructions to download and apply the required fixes for the integrated IBM WebSphere Process Server. Install the fixes only after you have updated IBM WebSphere Integration Developer V6.0.1 to Version 6.0.1.1.

> **Important:** You must apply the fixes to able to run the IMS sample applications in this redbook.

1. Point your browser to:

   `http://www-1.ibm.com/support/docview.wss?rs=180&context=SSEQTP&q1=websphere+update+installer&uid=swg24008401&loc=en_US&cs=utf-8&lang=en`

2. Select your platform (for example, Windows) and download the IBM UpdateInstaller to a temporary directory on your computer.

3. Extract the archive `updi.6000.windows.ia32.zip` to the WebSphere Integration Developer installation directory, for example:

   `C:\Program Files\IBM\Rational\SDP\6.0\runtimes\bi_v6`

   or

   `C:\Program Files\IBM\WebSphere\ID\6.0\runtimes\bi_v6`

4. Point your browser to:

   `http://www-1.ibm.com/support/docview.wss?uid=swg24011932`

5. Download the fix `APARJR23226`.
6. Switch to the `updateinstaller` directory and launch **update.exe**.
7. Click **Next**.

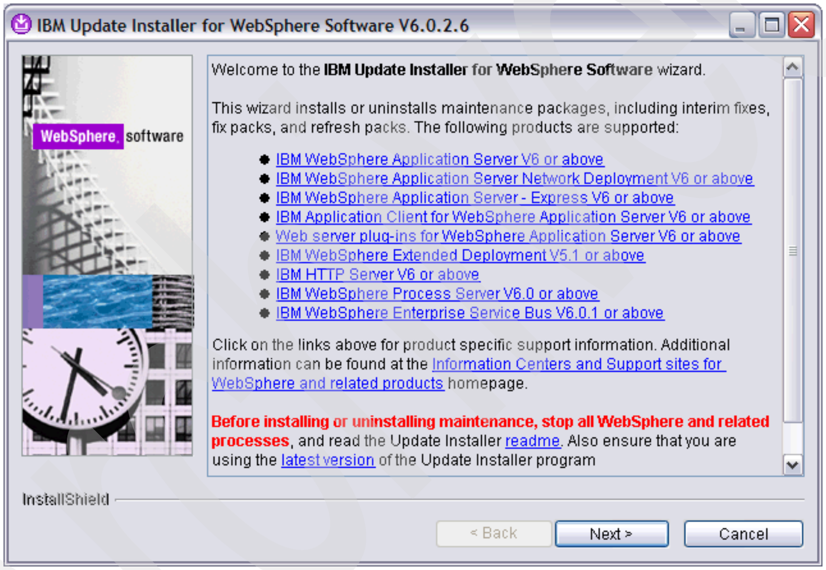


Figure A-26 The update installer welcome screen

8. Select the installation directory (accept the default) and click **Next**.
9. Select **Install maintenance package** and click **Next**.
10. Browse to the fixpack `6.0.1.0-WS-WPS-IF23226.pak` and click **Next**.
11. Confirm the selection.
12. Click **Next** to begin the installation.

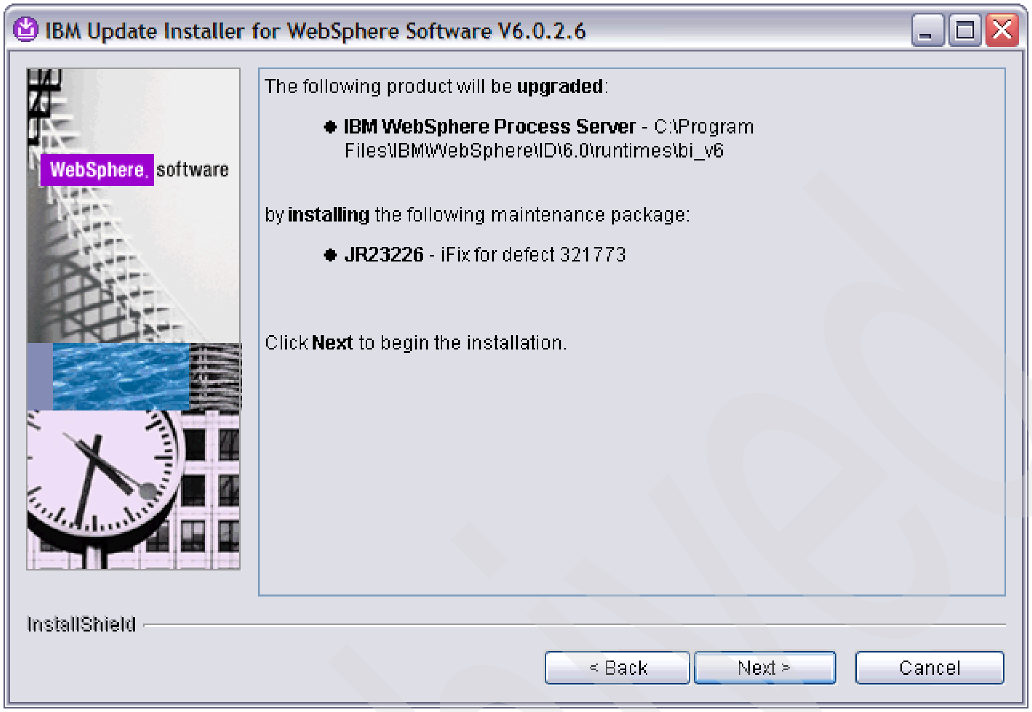


Figure A-27 The installation summary

13. Click **Finish** to exit the install process.

A.5 Installing the Telecom Web Services Server plug-in

In order to view and modify TWSS Mediation flows inside WebSphere Integration Developer it is necessary to install the Telecom Web Services Server plug-in and extract the ESB Mediation Flows.

1. Unzip the TWSS for Windows installation file into a temporary directory.
2. Switch to the `nodesetup` sub-directory.
3. Click **setup**. This starts the installation wizard for TWSS Node Configuration.
4. Click **Next**.
5. Click **Next** on the confirmation window.

Figure A-28 Start the TWSS Node Configuration wizard

6. Enter your WebSphere Application Server directory.

> **Note:** You need to have a WebSphere Application Server V6 installed.
>
> There are two options to choose from:
>
> 1. You already have a stand-alone WebSphere Application Server installation. Then choose the install directory, for example:
>
>   `C:/Program Files/IBM/WebSphere/AppServer`
>
> 2. You can select the directory of the integrated WebSphere Application Server that is part of for example the Rational Application Developer. In this case the installation directory would be similar to this one:
>
>   `C:/Program Files/IBM/Rational/SDP/6.0/runtimes/base_v6`

Figure A-29 Select WebSphere Application Server directory

3. Accept the confirmation of installation and click **Next**.
4. Click **Finish** to exit the wizard.
5. Copy the following directory from the `twss` sub-directory of your WebSphere Application Server installation directory to:
   `<WID_Install_Root>/eclipse/plugins`

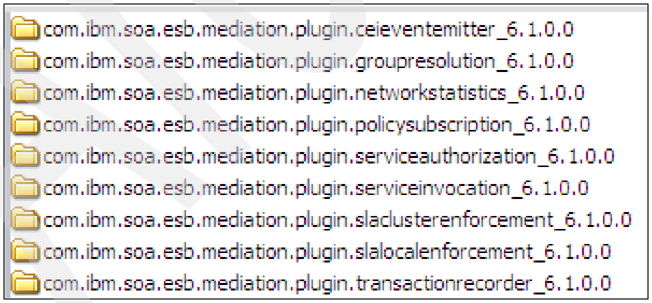


Figure A-30 Plug-in directories to import into WID

> **Attention:** To install the Eclipse plug-in properly you need to start WebSphere Integration Developer with the **`-clean`** option.

A.5.1 Extract the ESB mediation flows and import them into WID

Switch to the `esb` subdirectory from the temporary directory where you unzip the TWSS for Windows installation file into

From the directory where you unzipped the TWSS for Windows installation file:

1. Go to the `esb` sub-directory.
2. Click **setup**.
3. This starts the installation wizard. Click **Next**.

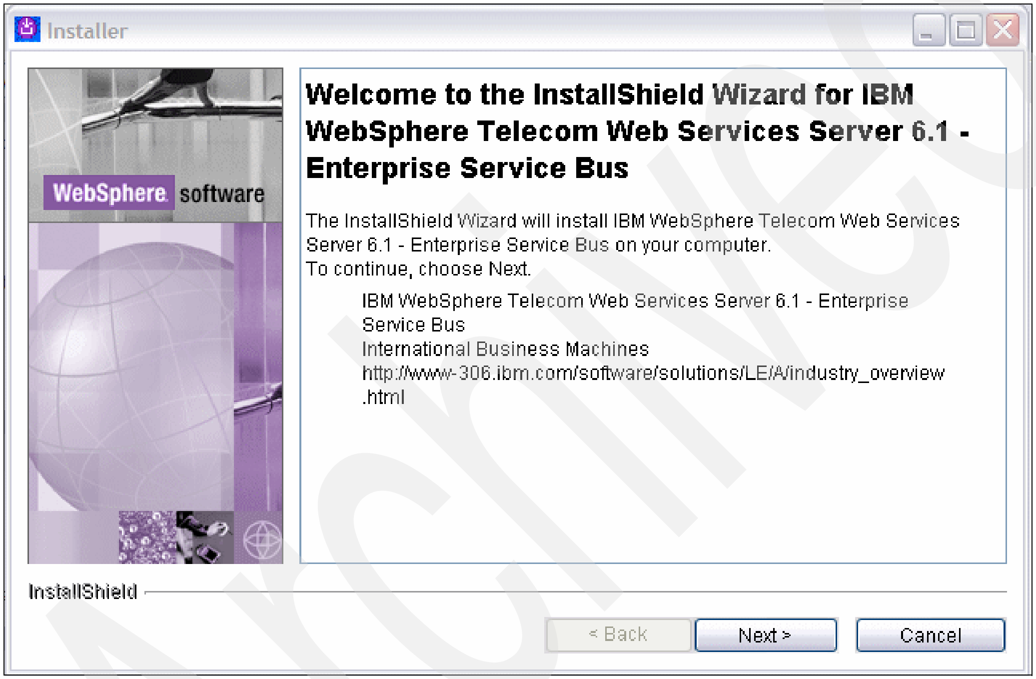


Figure A-31 TWSS Enterprise Service Bus installation wizard

4. Accept the licence agreement. Click **Next**.

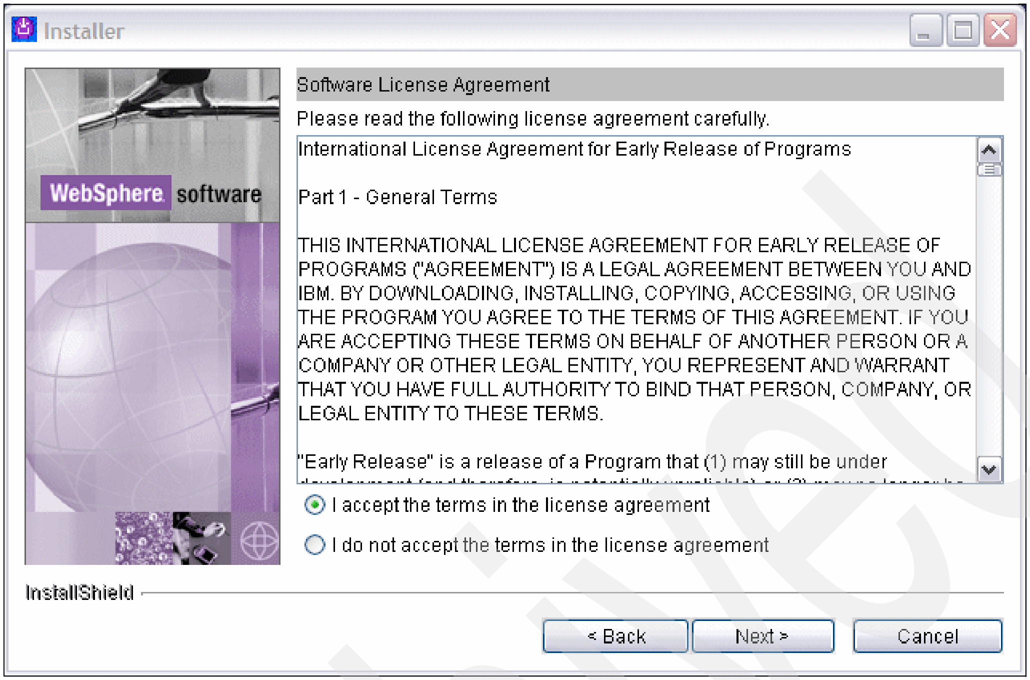


Figure A-32 Accept Licence Agreement

5. On the confirmation window, click **Next**.
6. Enter the directory where you want to install TWSS Enterprise Service Bus.

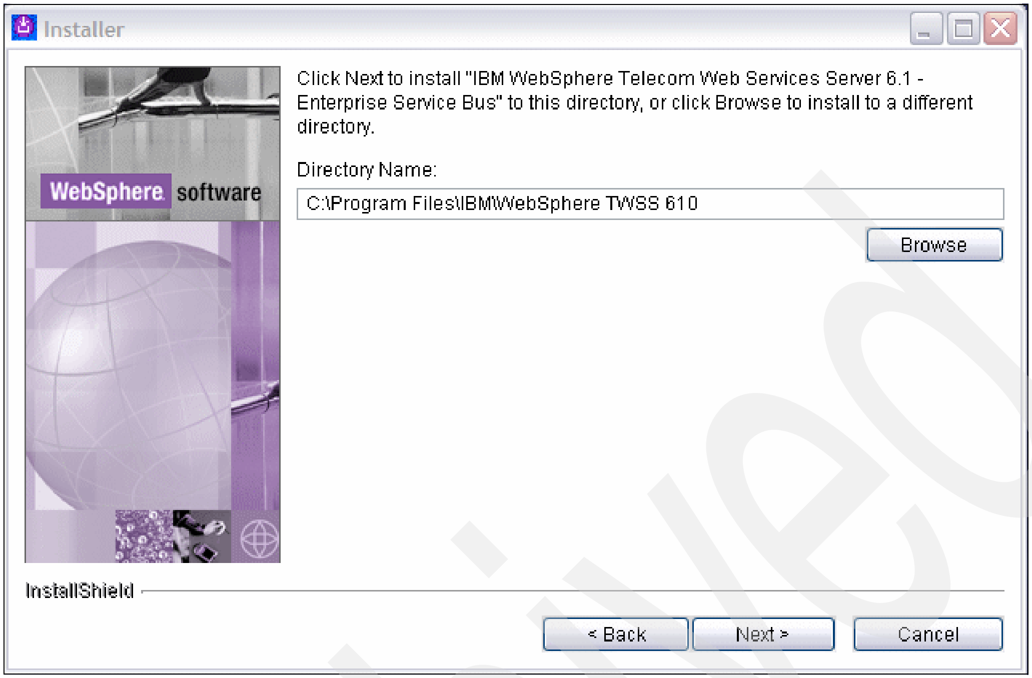


Figure A-33 Select installation directory

7. Accept the confirmation of installation. Click **Next**.
8. Click **Finish** to exit the wizard.
9. Switch back to the `esb` sub-directory of the place that you have just used for installation.
10. Copy the Parlay X mediation flows shown in Figure A-34 into: `<WID_install_Root>\runtimes\bi_v6\TWSS\ESB`

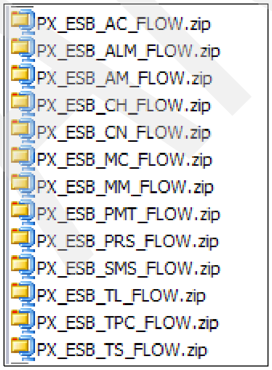


Figure A-34 Parlay X mediation flows

B

Installing the sample application test environment

This appendix provides the step-by-step description of the installation process for the sample application test environment.

This appendix contains the following:

- "IBM WebSphere Application Server 6.1"
- "IBM WebSphere Telecom Web Services Server"
- "IBM WebSphere Group List Server component"
- "IBM WebSphere Presence Server component"
- "IBM WebSphere Diameter Enabler component"

B.1 IBM WebSphere Application Server 6.1

This section describes the process for installing the WebSphere Application Server Network Deployment on both Linux and Windows operating systems.

> **Note:** The machine on which you install WebSphere Application Server must be running a supported distributed operating system. For more information about supported operating systems, see the WebSphere Application Server V6.1 Information Center.

The outline of the installation processes consists of preparing the install image and running the launchpad.

1. Create an install directory.
   - On Windows: `mkdir C:\temp\WAS61`
2. Copy the install image on the install directory.
   - For Windows

     Copy the **was.cd.6100.nd.windows.ia32.zip** install image into the **WAS61** directory.
   - For Linux

     Copy the **was.cd.6100.nd.linux.ia32.tar.gz** install image into the **WAS61** directory.
3. Unpack the install image.

   Navigate to the directory <temp>/WAS61.
   - For Windows

     Use the unzip utility to unzip the product image, for example:

     ```
     unzip ../was/cd.6100.nd.windows.ia32.zip
     ```
   - For Linux

     ```
     gzip -cd ../was.cd.6100. .nd.linux.ia32.tar.gz | tar -xvf
     ```
4. Start the WebSphere launchpad.
   a. On the Command Prompt navigate to the <temp>/WAS61 directory.
   b. Run launchpad.exe. The WebSphere Application Server Network Deployment Screen will appear.
5. Select **Launch the installation wizard for WebSphere Application Server Network Deployment**.

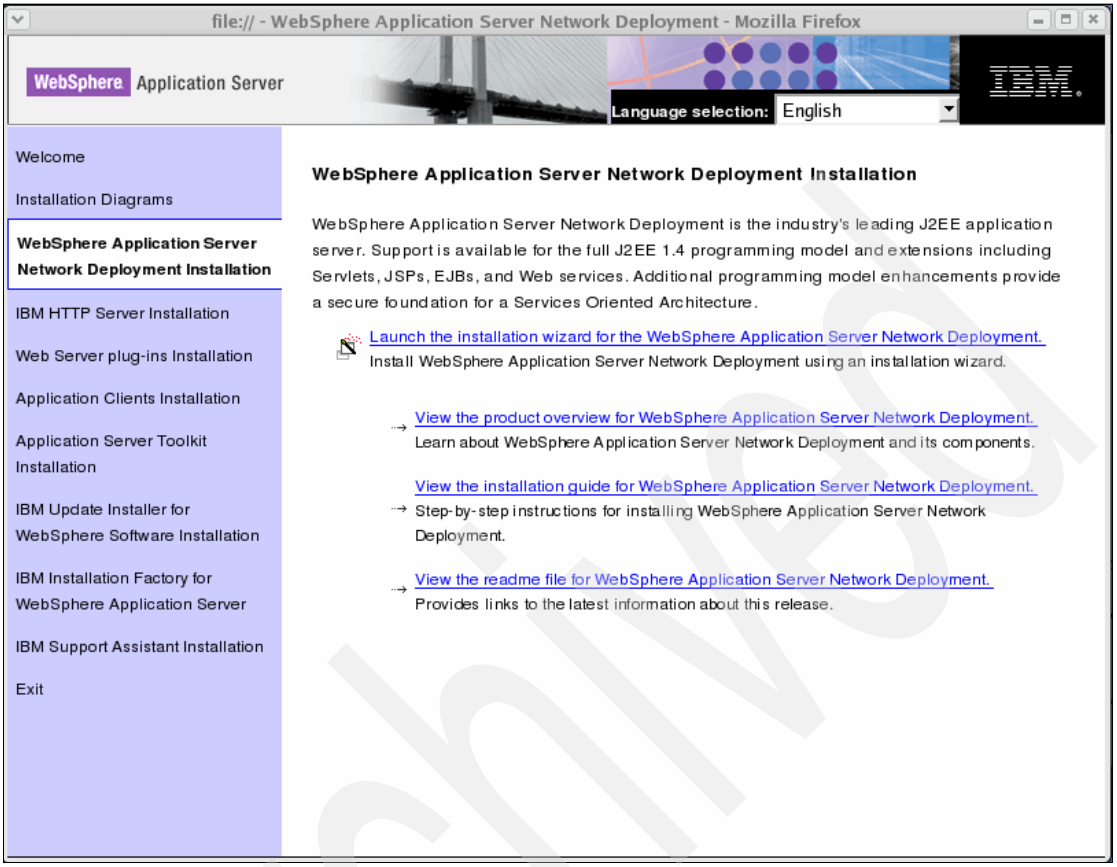


Figure B-1 WebSphere Application Server 6.1 Installation wizard

6. This starts the installation wizard for WebSphere Application Server Network Deployment installation.
7. Click **Next**.

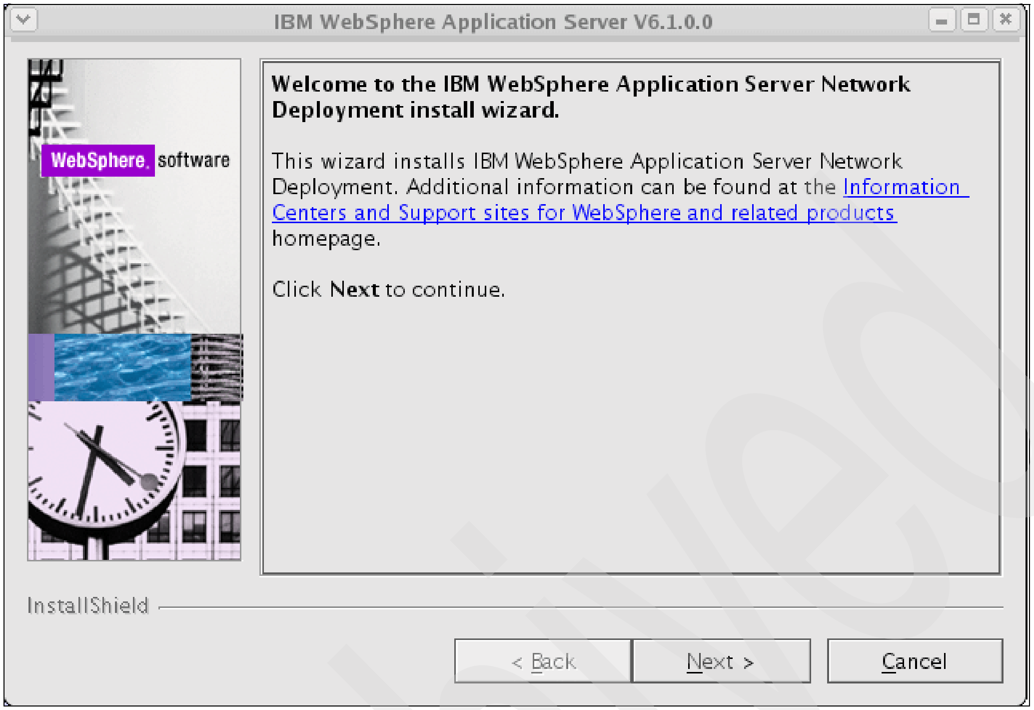


Figure B-2 Welcome pop-up

8. Accept the license agreement. Click **Next**.
9. The system prerequisites are checked. Then the features selection panel is displayed.
10. Check the **install of the sample applications** box.
11. Click **Next**.

> **Important:** You need to install these samples at this time, because you will not be able to do this later.

12. Select the installation location.
    - For Windows, type:

      `c:\WebSphere\AppServer as the installation path`
    - For Linux, type:

      `/opt/IBM/WebSphere/Appserver`
13. Click **Next**.

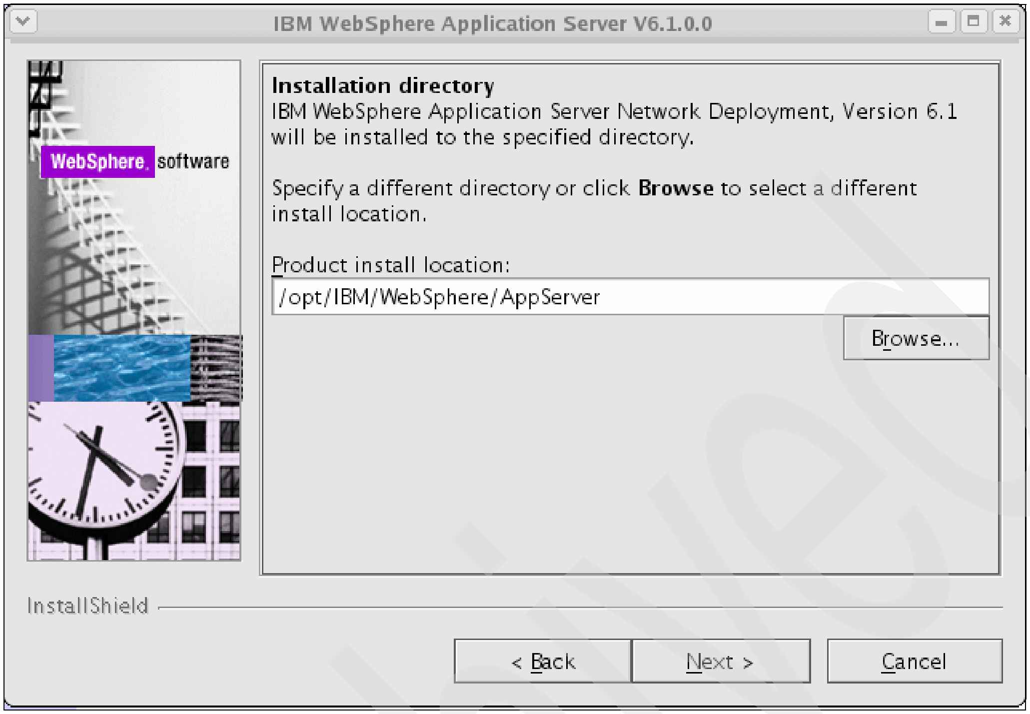


Figure B-3 WebSPhere Installation Directory

14.Next you choose the type of WebSphere Application Server environment to install. Choose **Application Server**.

> **Note:** For the purpose of running the sample application in this redbook, we installed a stand-alone server environment.

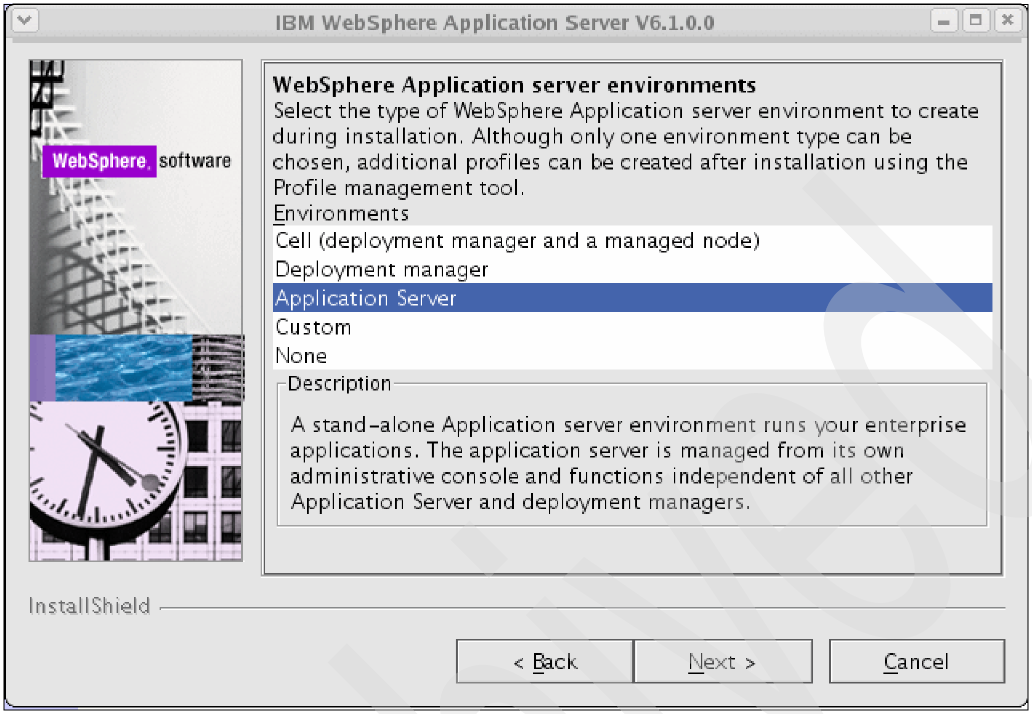


Figure B-4 WebSphere Application Server environments

An application server profile has a default server, named server1. The application server can be created with the default application and application samples installed.

15. Click **Next** to start the installation.

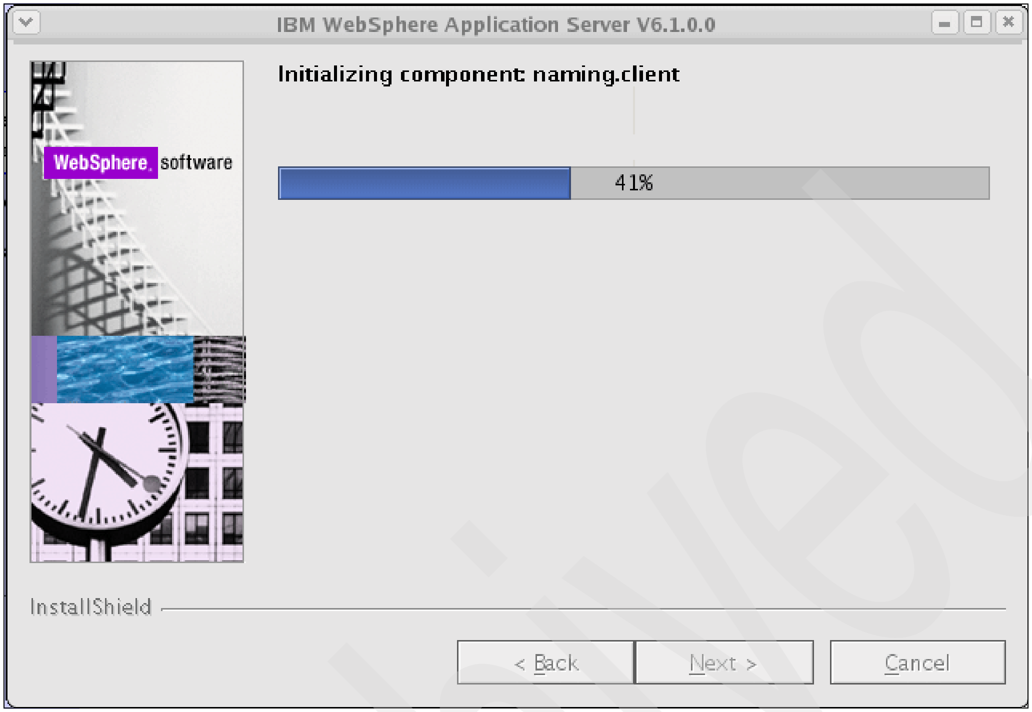


Figure B-5 WebSphere Installation in progress

16. When the installation is completed, verify that the status of the installation is **Success**.
17. Click **Finish**.

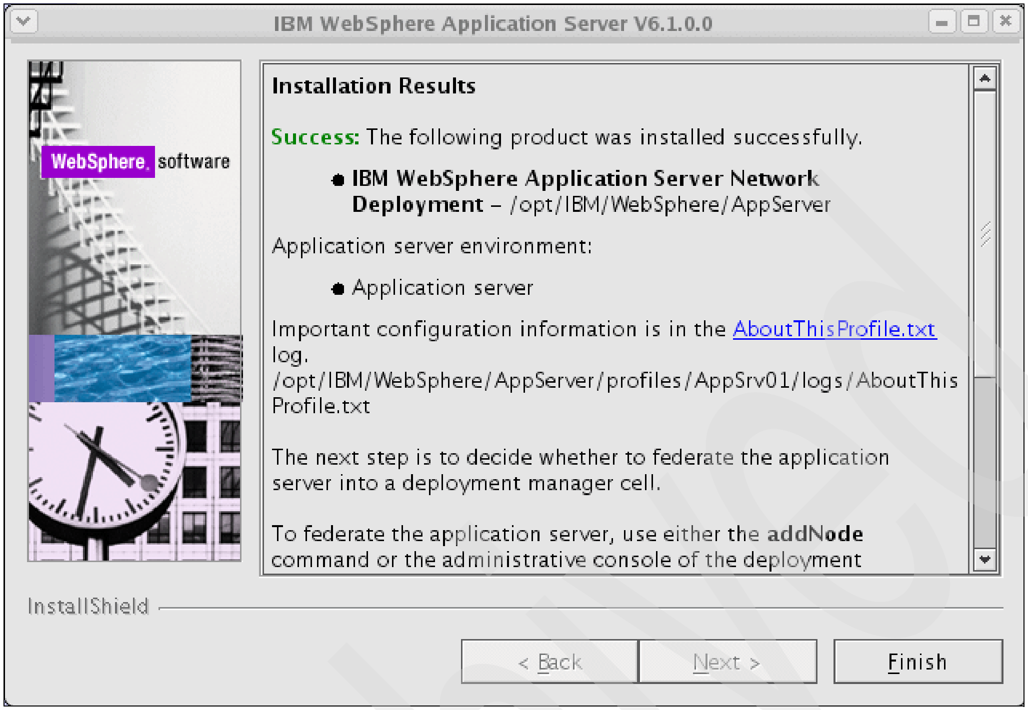


Figure B-6 Installation Results

18. The **First Steps** console will be displayed.
19. Select **Installation Verification**.

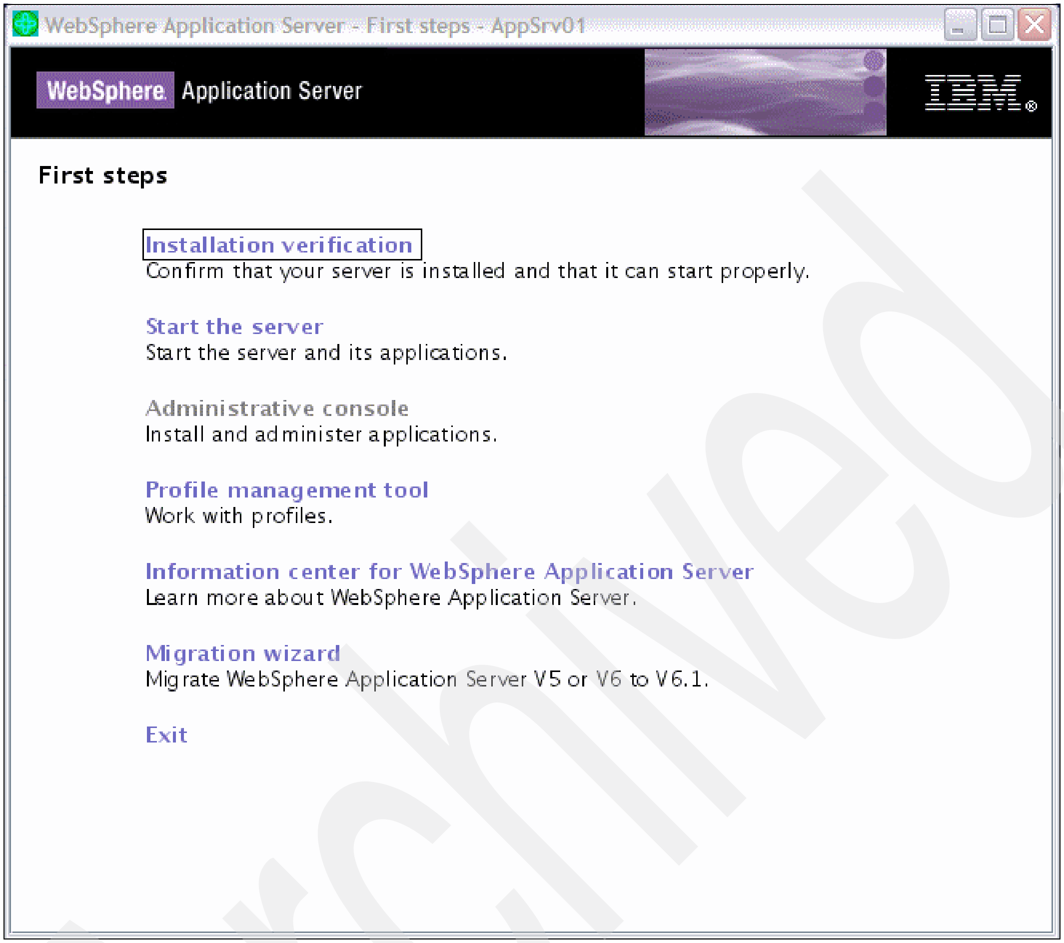


Figure B-7 First Steps console

20. Verify the output of the installation verification tool as in Figure B-8 on page 540. No errors should be detected.

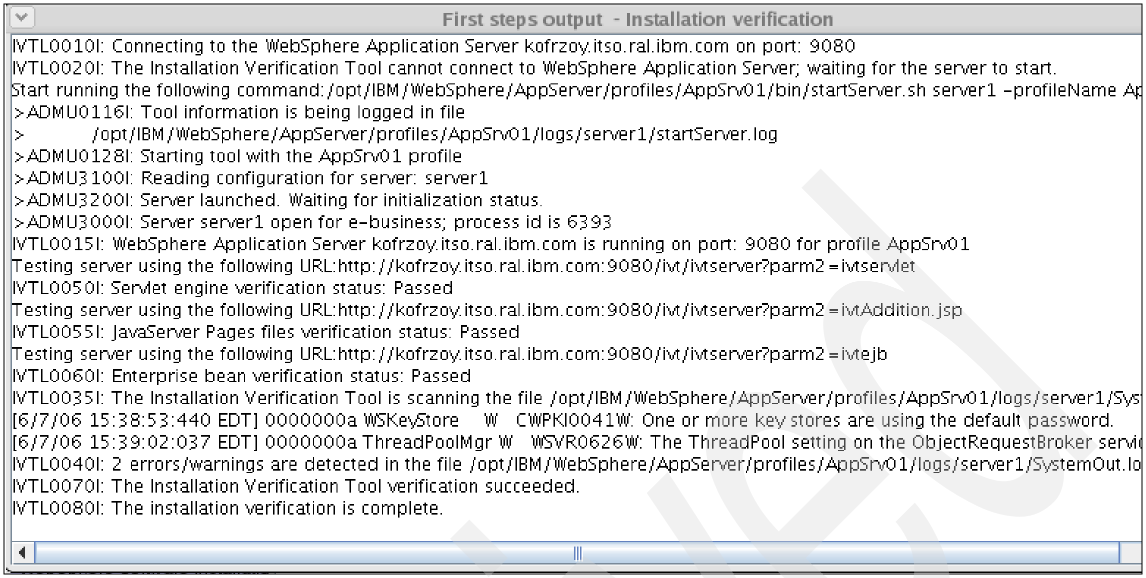


Figure B-8 Installation verification tool

B.2 IBM WebSphere Telecom Web Services Server

This section describes the steps required to install IBM WebSphere Telecom Web Services Server for the Third Party Call Control service.

The process describes the minimum installation required for executing the Third Party Call Control service in the sample applications. The outline of the installation processes consist of the following:

- B.2.1, "Create the WebSphere Application Server profile" on page 541
- B.2.2, "Install base binaries" on page 544
- B.2.3, "Configure DB2" on page 546
- B.2.4, "Configure Service Integration Bus" on page 554
- B.2.5, "Configure JDBC" on page 561
- B.2.6, "Tune the Application Server" on page 567
- B.2.7, "Deploy TWSS Applications" on page 570
- B.2.8, "Verify the installation" on page 582
- B.2.9, "Troubleshoot the installation" on page 583

If you need additional functionality, refer to the IBM WebSphere Telecom Web Services Server Information Center for additional installation steps.

Note: The installation instructions assume that WebSphere Application Server 6.1 and DB2 are already installed and configured on the machine. If you require detailed instructions regarding the installation of these products, consult the following resources:

- DB2 Universal Database for Linux, UNIX® and Windows Information Center

  `http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp`

- WebSphere Application Server 6.1 Information Center

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp`

B.2.1 Create the WebSphere Application Server profile

The sample test environment is composed of different WebSphere Application Server for the IBM IP Multimedia Subsystem components (Presence Server, Group List Server, IMS Connector and TWSS). Creating a new application profile removes potential conflicts between the components for resources and also allows performance tuning of the Java Virtual Machine on a product basis.

1. Switch to <was_root>/firststeps directory.
2. Ensure that the DISPLAY variable is correctly defined:

   ```
   export DISPLAY=9.1.2.3:0
   ```

   Where, 9.1.2.3 is the IP address or hostname of the system where the GUI will be displayed.
3. Start the First steps GUI:

   ```
   ./firststeps.sh
   ```

4. In the First steps menu, click **Profile management tool**, as in Figure B-9 on page 542.

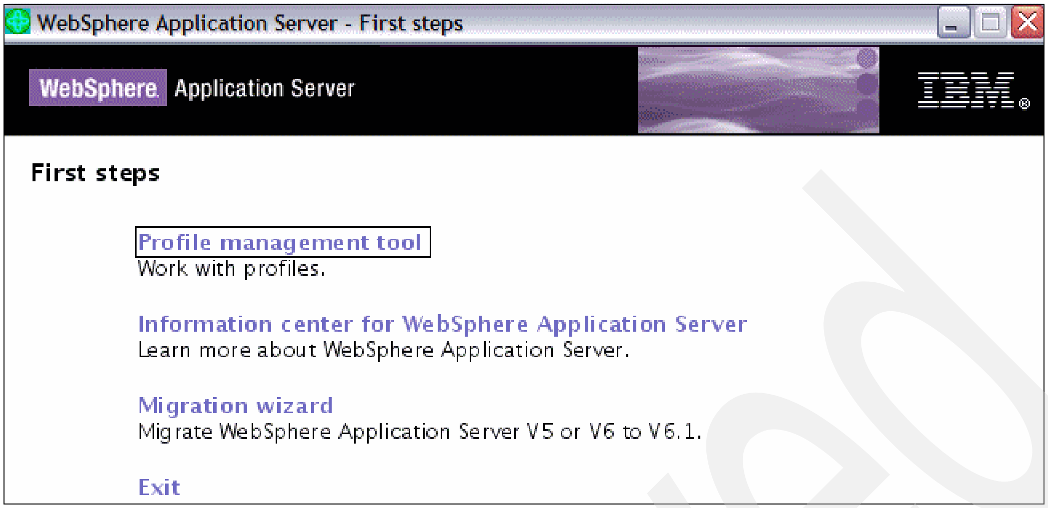


Figure B-9 First steps menu

5. The Profile Management Tool will appear. Click **Next**.
6. In the Environment Selection window, select **Application server** from the environments, and then click **Next**.

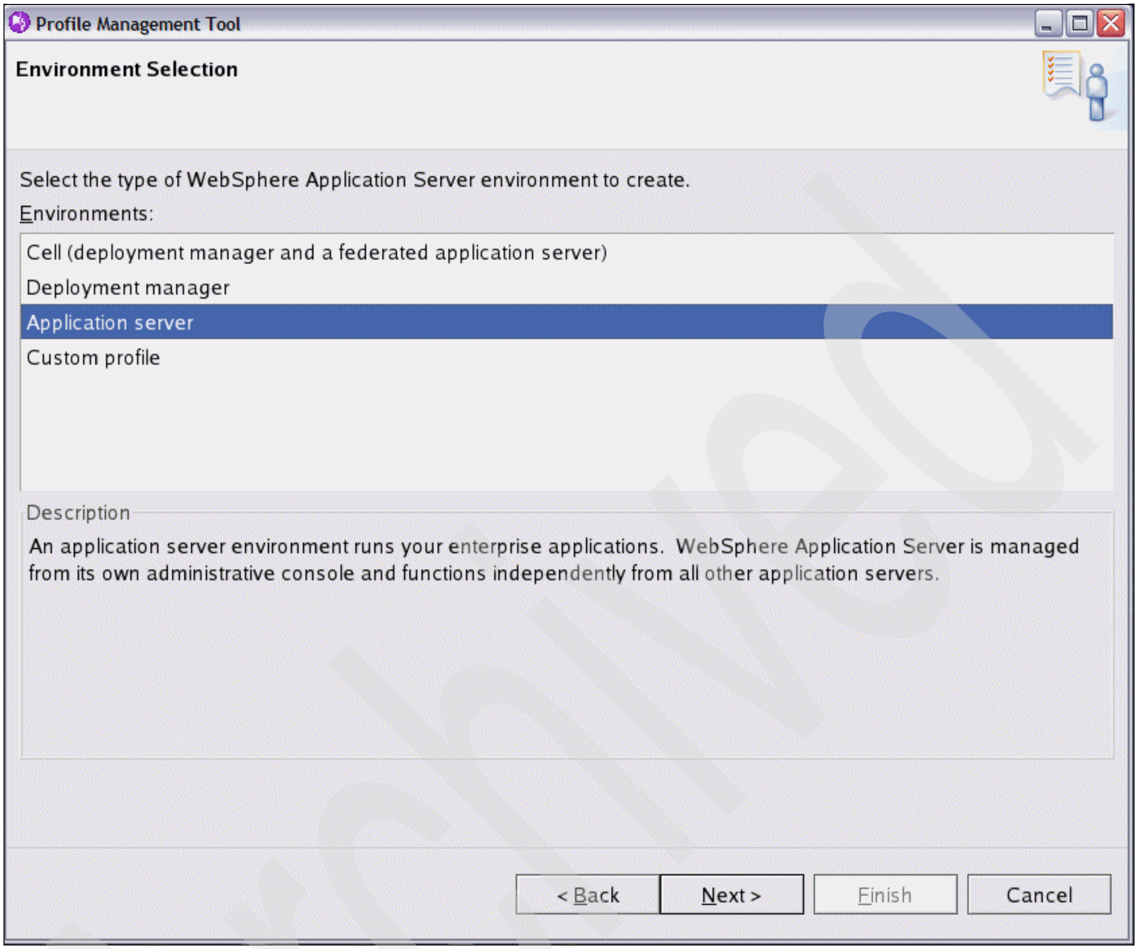


Figure B-10 Environment selection for the Profile Management Tool

7. For the Profile Creation Options, select **Typical Profile Creation**, and then click **Next**.
8. Enable administrative security.
   a. Check the **Enable administrative security** box.
   b. Enter the **User name** and **Password** for the new Application Server.
   c. Click **Next**.

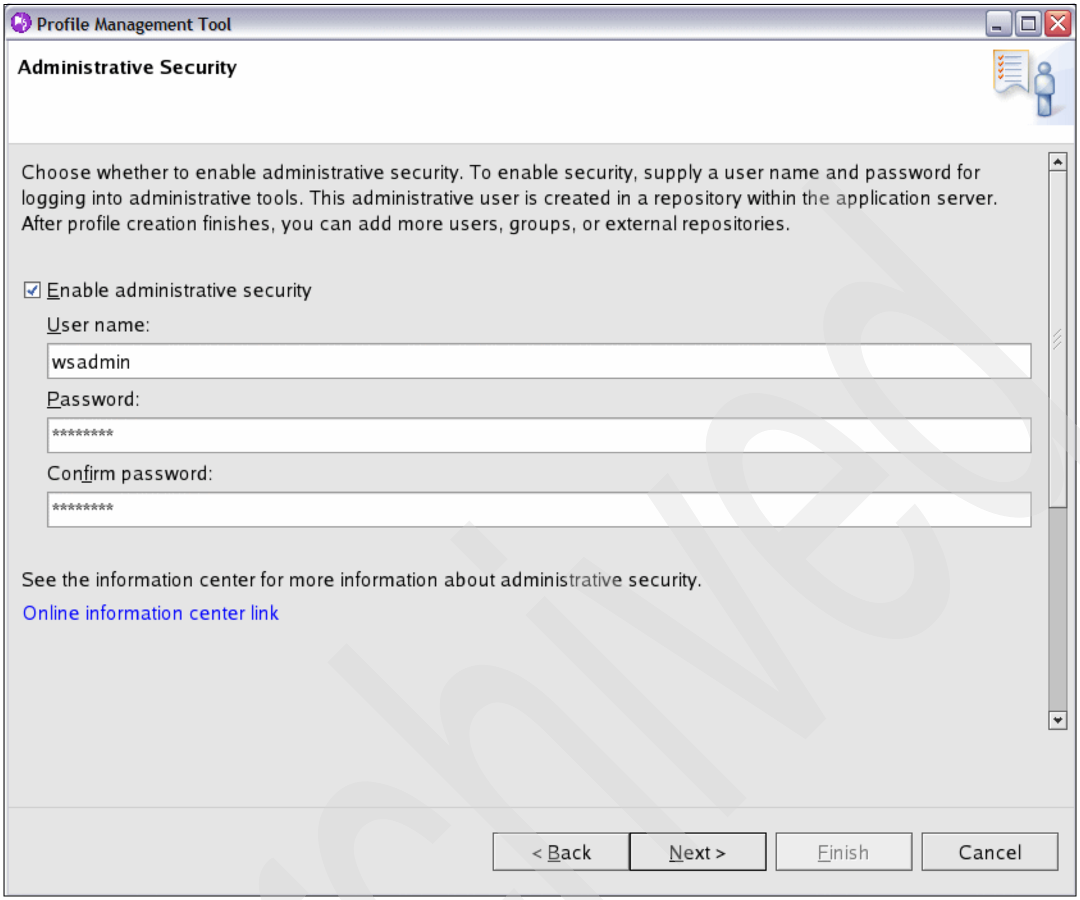


Figure B-11 Enabling Administration Security

9. The summary of the new Profile is shown. Click **Create**.
10. The profile is created and the completion screen will appear.
    a. Deselect all options.
    b. Click **Finish** to close the Profile Management Tool.

B.2.2 Install base binaries

TWSS has several binaries available to install. Different binaries are installed depending on the functionality required. Each binary installation process is based on the same menu structure with slightly different values for the installation and setup path. Table B-1 lists the properties for installing the different binaries.

Table B-1 TWSS Components and settings for installation

Component Name	Setup File Location	Installation Directory
ESB extensions	esb	/opt/IBM/TWSS
Web Services	services	/opt/IBM/TWSS
Administration Console	adminconsole	/opt/IBM/WebSphere/AppServer /systemApps/isclite.ear/iclite.war
Node Configuration	nodesetup	/opt/IBM/WebSphere/AppServer
Service Platform	serviceplatform	/opt/IBM/TWSS

1. Login to the system as root.
2. Ensure that the DISPLAY variable is correctly defined:

   ```
   export DISPLAY=9.1.2.3:0
   ```

   Where, 9.1.2.3 is the IP address or hostname of the system where the GUI will be displayed.
3. Change to the setup file location as specified in Table B-1.
4. Run the installation wizard:

   ```
   ./setup
   ```

   Select a language. Click **OK**.
5. In the Welcome window, click **Next**.
6. Accept the License Agreement.
   a. Read the license agreement carefully.
   b. Tick the check box to accept.
   c. Click **Next**.
7. Click **Next**.
8. Specify the installation destination directory.
   a. Modify the directory installation field to correspond to the values in Table B-1.
   b. Click **Next**.

Figure B-12 Binary installation directory

9. Once the installation has completed, a summary of the installation settings will be displayed. Click **Next**.
10. In the completion window, click **Finish** to close the installation wizard.

> **Note:** Repeat the steps above starting from Step 3 for installing the different components contained in Table B-1 on page 545.

B.2.3 Configure DB2

TWSS utilizes database tables for storing configuration, runtime, and logging information. Components of the IBM WebSphere products for telecommunication are designed to use DB2 or Oracle databases. The process presented here describes configuration for DB2 only.

> **Note:** It assumes that DB2 8.2 fix pack 4 has been installed. If you require detailed instructions regarding the installation, consult the following resource:
>
> - DB2 Universal Database for Linux, UNIX and Windows Information Center
>
>   `http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp`

Similar to the installation process in “Install base binaries” on page 544 different tables are installed depending on the functionality used in TWSS. DB2 databases and tables are created using the script crtsrvDb2.sh. Multiple versions of this scripts were created for the installation of the TWSS binaries. These will be used for different sections of the database configuration. The invocation mechanism is common across all versions of the script. The scripts are invoked using twelve arguments which modify the behavior of the script execution.

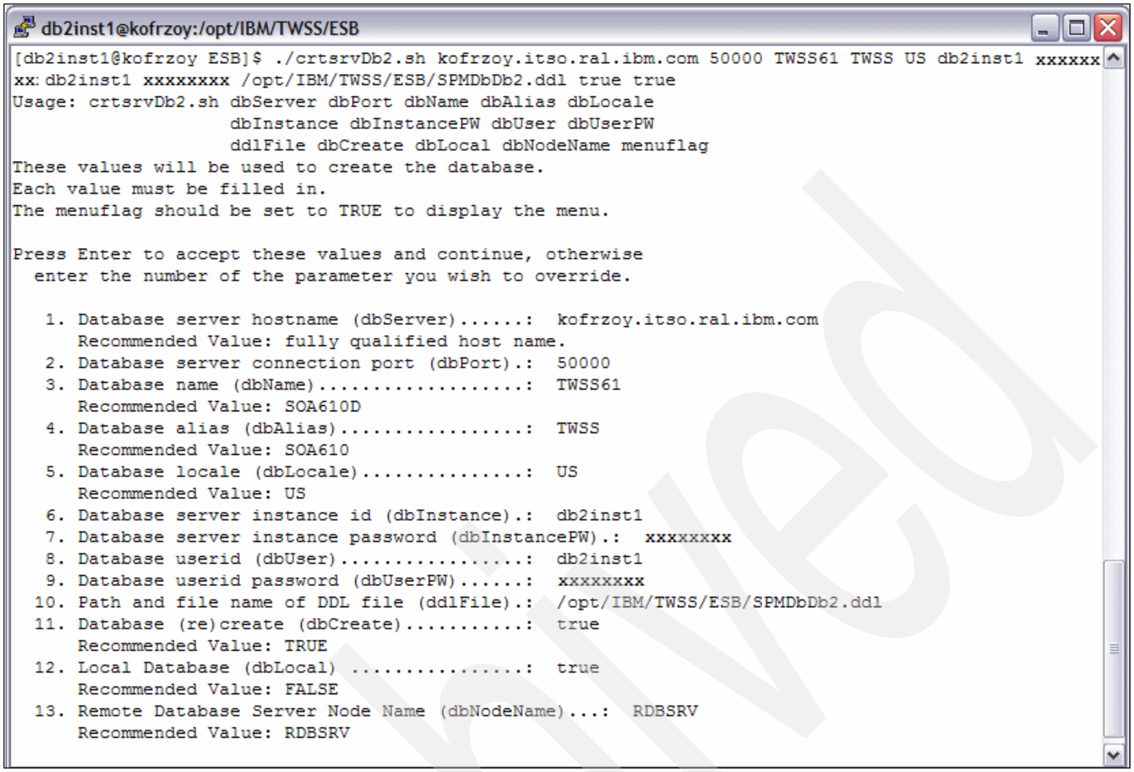
Table B-2 Script arguments

Argument	Description
Database server hostname	This value must be the hostname and domain (for example machine1.ibm.com) **Note:** Localhost should not be used even if the database is hosted locally.
Database server connection port	This value is normally `50000`
Database name	This can be any valued DB2 database name, however `TWSS610` was used for this example
Database alias	This can be any valued DB2 database alias, however `TWSS` was used for this example
Database locale	For this sample test environment, this value was set to `US`
Database server instance ID	In this example, this value was set to `db2inst1`
Database server instance password	
Database user ID	In this example, the user ID was set to `db2inst1`
Database user ID password	
Path and file name of DDL file	This value will change depending on the request that is being made
Database (re)create	This determines if the script will drop the database, and recreate at the beginning of the script.
Local Database	Is the database that will be used for the TWSS hosted locally
Remote Database Server Node Name	This is not appropriate if the database is being hosted locally

1. Log into the system as root.

2. Change/modify permissions.
   a. Change the permission of /opt/IBM/TWSS/ESB/crtsrvDb2.sh:
   ```
   chmod 755 /opt/IBM/TWSS/ESB/crtsrvDb2.sh
   ```
   b. Modify the permission of /opt/IBM/TWSS/ServicePlatform/crtsrvDb2.sh:
   ```
   chmod 755 /opt/IBM/TWSS/ServicePlatform/crtsrvDb2.sh
   ```
   c. Modify the permission of /opt/IBM/TWSS/IMSServices/crtsrvDb2.sh:
   ```
   chmod 755 /opt/IBM/TWSS/IMSServices/crtsrvDb2.sh
   ```
3. Switch to a user ID with database administrator authority, such as db2inst1:
   ```
   su - db2inst1
   ```
4. Change directory to the location of the ESB binaries:
   ```
   cd /opt/IBM/TWSS/ESB
   ```
5. Execute the following command:
   ```
   ./crtsrvDb2.sh <fully qualified hostname> 50000 TWSS61 TWSS US
   db2inst1 xxxxxxxx db2inst1 xxxxxxxx /opt/IBM/TWSS/ESB/SPMDbDb2.ddl
   true true
   ```
   Press Enter twice to create the database configuration.

> **Note:** The xxxxxxxx and xxxxxxxx is the database server instance password and database userid password, respectively, in:
>
> ```
> ./crtsrvDb2.sh <fully qualified hostname> 50000 TWSS61 TWSS US
> db2inst1 xxxxxxxx db2inst1 xxxxxxxx /opt/IBM/TWSS/ESB/SPMDbDb2.ddl
> true true
> ```



```
db2inst1@kofrzoy:/opt/IBM/TWSS/ESB
[db2inst1@kofrzoy ESB]$ ./crtsrvDb2.sh kofrzoy.itso.ral.ibm.com 50000 TWSS61 TWSS US db2inst1 xxxxxx
xx: db2inst1 xxxxxxxx /opt/IBM/TWSS/ESB/SPMDbDb2.ddl true true
Usage: crtsrvDb2.sh dbServer dbPort dbName dbAlias dbLocale
                    dbInstance dbInstancePW dbUser dbUserPW
                    ddlFile dbCreate dbLocal dbNodeName menuflag
These values will be used to create the database.
Each value must be filled in.
The menuflag should be set to TRUE to display the menu.

Press Enter to accept these values and continue, otherwise
  enter the number of the parameter you wish to override.

   1. Database server hostname (dbServer)......:  kofrzoy.itso.ral.ibm.com
      Recommended Value: fully qualified host name.
   2. Database server connection port (dbPort).:  50000
   3. Database name (dbName)...................:  TWSS61
      Recommended Value: SOA610D
   4. Database alias (dbAlias).................:  TWSS
      Recommended Value: SOA610
   5. Database locale (dbLocale)...............:  US
      Recommended Value: US
   6. Database server instance id (dbInstance).:  db2inst1
   7. Database server instance password (dbInstancePW).:  xxxxxxxx
   8. Database userid (dbUser).................:  db2inst1
   9. Database userid password (dbUserPW)......:  xxxxxxxx
  10. Path and file name of DDL file (ddlFile).:  /opt/IBM/TWSS/ESB/SPMDbDb2.ddl
  11. Database (re)create (dbCreate)...........:  true
      Recommended Value: TRUE
  12. Local Database (dbLocal) ................:  true
      Recommended Value: FALSE
  13. Remote Database Server Node Name (dbNodeName)...:  RDBSRV
      Recommended Value: RDBSRV
```

Figure B-13 Installation of the Service Policy Manager Database configuration

6. Execute the following command:

```
./crtsrvDb2.sh <fully qualified hostname> 50000 TWSS61 TWSS US
db2inst1 xxxxxxxx db2inst1 xxxxxxxx /opt/IBM/TWSS/ESB/ESBDbDb2.ddl
false true
```

   Press Enter twice to create the database configuration.

```
db2inst1@kofrzoy:/opt/IBM/TWSS/ESB
[db2inst1@kofrzoy ESB]$ ./crtsrvDb2.sh kofrzoy.itso.ral.ibm.com 50000 TWSS61 TWSS US db2inst1 xxxxxx
xx db2inst1 xxxxxxxx /opt/IBM/TWSS/ESB/ESBDbDb2.ddl false true
Usage: crtsrvDb2.sh dbServer dbPort dbName dbAlias dbLocale
                    dbInstance dbInstancePW dbUser dbUserPW
                    ddlFile dbCreate dbLocal dbNodeName menuflag
These values will be used to create the database.
Each value must be filled in.
The menuflag should be set to TRUE to display the menu.

Press Enter to accept these values and continue, otherwise
  enter the number of the parameter you wish to override.

   1. Database server hostname (dbServer)......:  kofrzoy.itso.ral.ibm.com
      Recommended Value: fully qualified host name.
   2. Database server connection port (dbPort).:  50000
   3. Database name (dbName)...................:  TWSS61
      Recommended Value: SOA610D
   4. Database alias (dbAlias).................:  TWSS
      Recommended Value: SOA610
   5. Database locale (dbLocale)...............:  US
      Recommended Value: US
   6. Database server instance id (dbInstance).:  db2inst1
   7. Database server instance password (dbInstancePW).:  xxxxxxxx
   8. Database userid (dbUser).................:  db2inst1
   9. Database userid password (dbUserPW)......:  xxxxxxxx
  10. Path and file name of DDL file (ddlFile).:  /opt/IBM/TWSS/ESB/ESBDbDb2.ddl
  11. Database (re)create (dbCreate)...........:  false
      Recommended Value: TRUE
  12. Local Database (dbLocal) ................:  true
      Recommended Value: FALSE
  13. Remote Database Server Node Name (dbNodeName)...:  RDBSRV
      Recommended Value: RDBSRV
```

Figure B-14 Installation of the ESB Database configuration

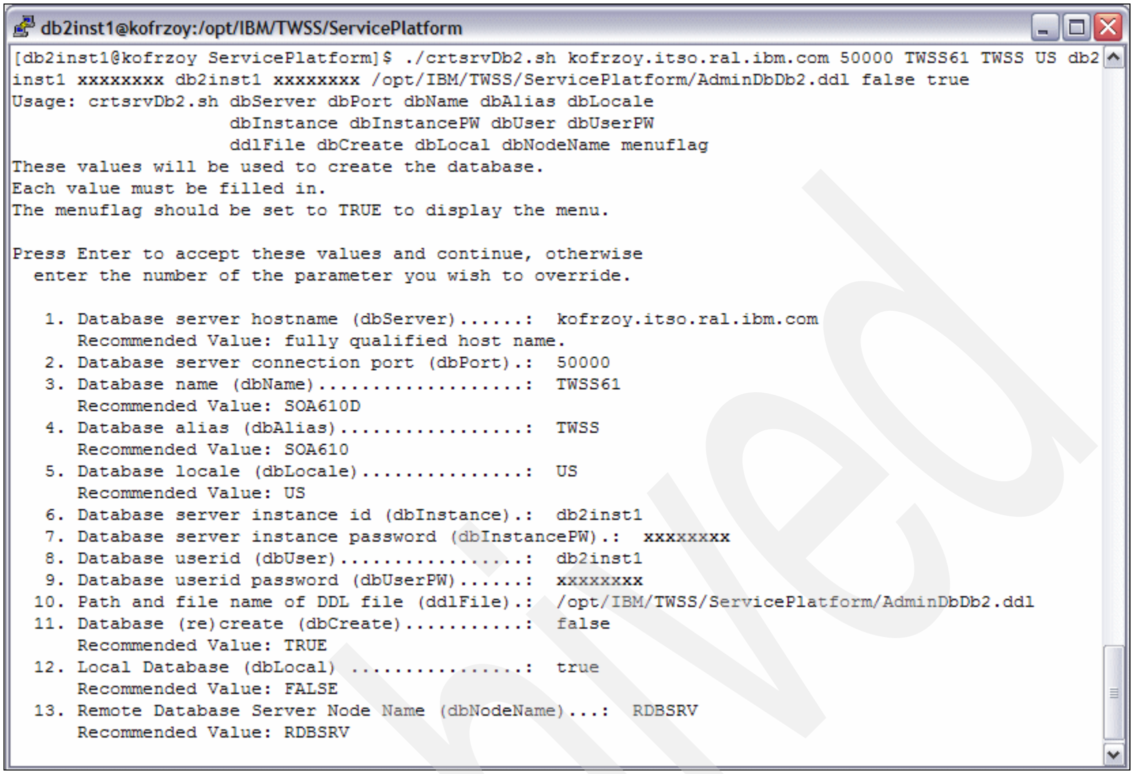
7. Change directory to the location of the Service Platform binaries:

   ```
   cd /opt/IBM/TWSS/ServicePlatform
   ```

8. Execute the following command:

   ```
   ./crtsrvDb2.sh <fully qualified hostname> 50000 TWSS61 TWSS US
   db2inst1 xxxxxxxx db2inst1 xxxxxxxx
   /opt/IBM/TWSS/ServicePlatform/AdminDbDb2.ddl false true
   ```

   Press Enter twice to create the database configuration.



```
db2inst1@kofrzoy:/opt/IBM/TWSS/ServicePlatform
[db2inst1@kofrzoy ServicePlatform]$ ./crtsrvDb2.sh kofrzoy.itso.ral.ibm.com 50000 TWSS61 TWSS US db2
inst1 xxxxxxxx db2inst1 xxxxxxxx /opt/IBM/TWSS/ServicePlatform/AdminDbDb2.ddl false true
Usage: crtsrvDb2.sh dbServer dbPort dbName dbAlias dbLocale
                    dbInstance dbInstancePW dbUser dbUserPW
                    ddlFile dbCreate dbLocal dbNodeName menuflag
These values will be used to create the database.
Each value must be filled in.
The menuflag should be set to TRUE to display the menu.

Press Enter to accept these values and continue, otherwise
  enter the number of the parameter you wish to override.

   1. Database server hostname (dbServer)......:  kofrzoy.itso.ral.ibm.com
      Recommended Value: fully qualified host name.
   2. Database server connection port (dbPort).:  50000
   3. Database name (dbName)...................:  TWSS61
      Recommended Value: SOA610D
   4. Database alias (dbAlias).................:  TWSS
      Recommended Value: SOA610
   5. Database locale (dbLocale)...............:  US
      Recommended Value: US
   6. Database server instance id (dbInstance).:  db2inst1
   7. Database server instance password (dbInstancePW).:  xxxxxxxx
   8. Database userid (dbUser).................:  db2inst1
   9. Database userid password (dbUserPW)......:  xxxxxxxx
  10. Path and file name of DDL file (ddlFile).:  /opt/IBM/TWSS/ServicePlatform/AdminDbDb2.ddl
  11. Database (re)create (dbCreate)...........:  false
      Recommended Value: TRUE
  12. Local Database (dbLocal) ................:  true
      Recommended Value: FALSE
  13. Remote Database Server Node Name (dbNodeName)...:  RDBSRV
      Recommended Value: RDBSRV
```

Figure B-15 Installation of the Administration Console Database configuration

9. Execute the following command:

```
./crtsrvDb2.sh <fully qualified hostname> 50000 TWSS61 TWSS US
db2inst1 xxxxxxxx db2inst1 xxxxxxxx
/opt/IBM/TWSS/ServicePlatform/UsageDbDb2.ddl false true
```

Press Enter twice to create the database configuration.

```
[db2inst1@kofrzoy ServicePlatform]$ ./crtsrvDb2.sh kofrzoy.itso.ral.ibm.com 50000 TWSS61 TWSS US db2
inst1 xxxxxxxx db2inst1 xxxxxxxx /opt/IBM/TWSS/ServicePlatform/UsageDbDb2.ddl false true
Usage: crtsrvDb2.sh dbServer dbPort dbName dbAlias dbLocale
                    dbInstance dbInstancePW dbUser dbUserPW
                    ddlFile dbCreate dbLocal dbNodeName menuflag
These values will be used to create the database.
Each value must be filled in.
The menuflag should be set to TRUE to display the menu.

Press Enter to accept these values and continue, otherwise
  enter the number of the parameter you wish to override.

   1. Database server hostname (dbServer)......:  kofrzoy.itso.ral.ibm.com
      Recommended Value: fully qualified host name.
   2. Database server connection port (dbPort).:  50000
   3. Database name (dbName)...................:  TWSS61
      Recommended Value: SOA610D
   4. Database alias (dbAlias).................:  TWSS
      Recommended Value: SOA610
   5. Database locale (dbLocale)...............:  US
      Recommended Value: US
   6. Database server instance id (dbInstance).:  db2inst1
   7. Database server instance password (dbInstancePW).:  xxxxxxxx
   8. Database userid (dbUser).................:  db2inst1
   9. Database userid password (dbUserPW)......:  xxxxxxxx
  10. Path and file name of DDL file (ddlFile).:  /opt/IBM/TWSS/ServicePlatform/UsageDbDb2.ddl
  11. Database (re)create (dbCreate)...........:  false
      Recommended Value: TRUE
  12. Local Database (dbLocal) ................:  true
      Recommended Value: FALSE
  13. Remote Database Server Node Name (dbNodeName)...:  RDBSRV
      Recommended Value: RDBSRV
```

Figure B-16 Installation of the Usage Database configuration

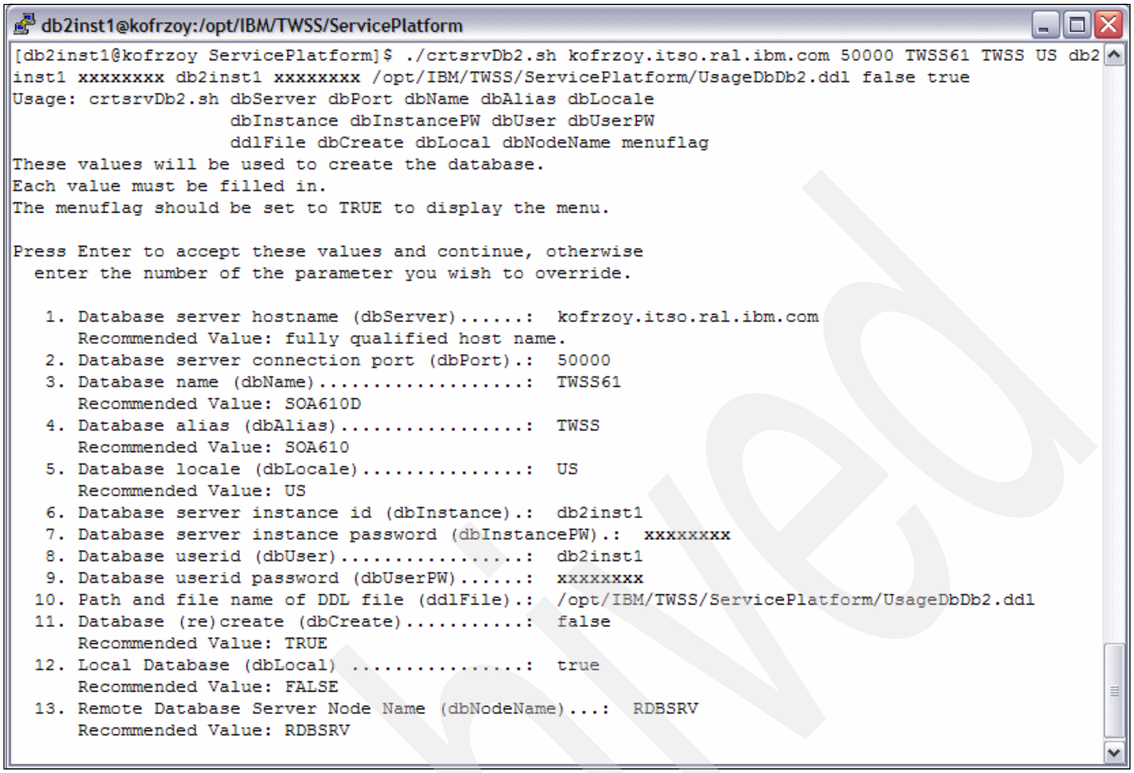
10. Change directory to the location of the IMSServices binaries:

    cd /opt/IBM/TWSS/IMSServices

11. Execute the following command:

    ./crtsrvDb2.sh <fully qualified hostname> 50000 TWSS61 TWSS US db2inst1 xxxxxxxx db2inst1 xxxxxxxx /opt/IBM/TWSS/IMSServices/ThirdPartyDbDb2.ddl false true

    Press Enter twice to create the database configuration.



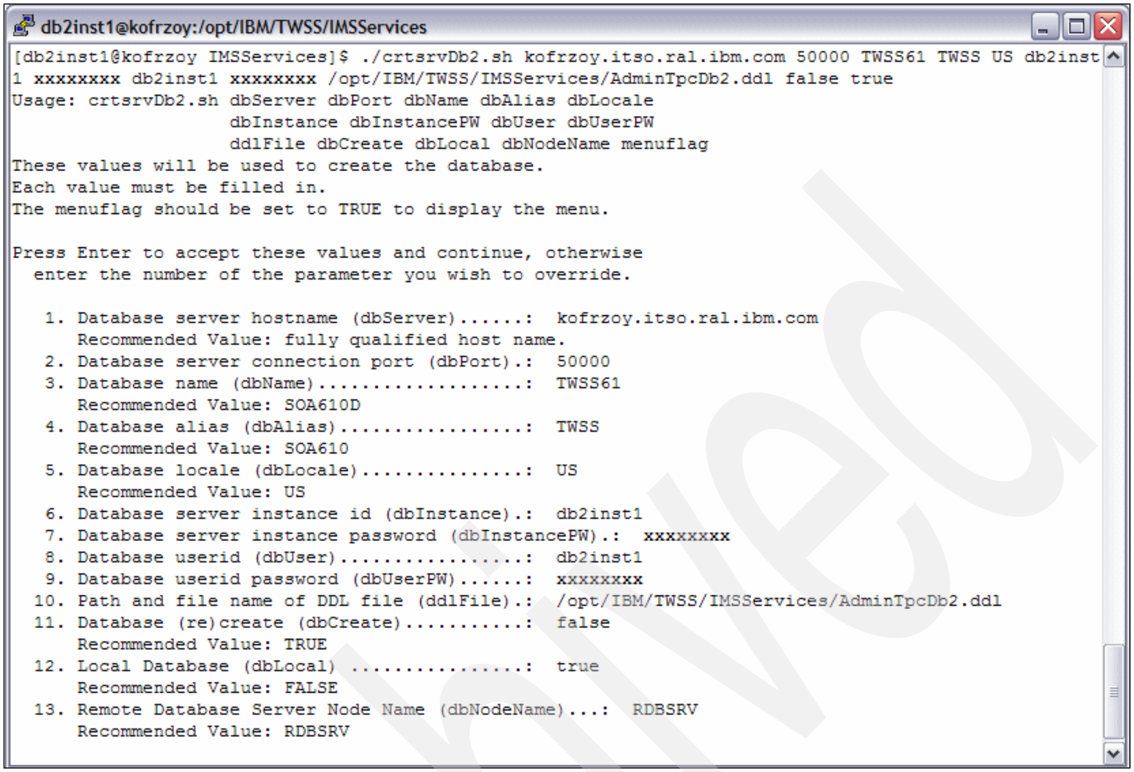
```
db2inst1@kofrzoy:/opt/IBM/TWSS/ServicePlatform
[db2inst1@kofrzoy ServicePlatform]$ ./crtsrvDb2.sh kofrzoy.itso.ral.ibm.com 50000 TWSS61 TWSS US db2
inst1 xxxxxxxx db2inst1 xxxxxxxx /opt/IBM/TWSS/ServicePlatform/UsageDbDb2.ddl false true
Usage: crtsrvDb2.sh dbServer dbPort dbName dbAlias dbLocale
                    dbInstance dbInstancePW dbUser dbUserPW
                    ddlFile dbCreate dbLocal dbNodeName menuflag
These values will be used to create the database.
Each value must be filled in.
The menuflag should be set to TRUE to display the menu.

Press Enter to accept these values and continue, otherwise
  enter the number of the parameter you wish to override.

   1. Database server hostname (dbServer)......:  kofrzoy.itso.ral.ibm.com
      Recommended Value: fully qualified host name.
   2. Database server connection port (dbPort).:  50000
   3. Database name (dbName)...................:  TWSS61
      Recommended Value: SOA610D
   4. Database alias (dbAlias).................:  TWSS
      Recommended Value: SOA610
   5. Database locale (dbLocale)...............:  US
      Recommended Value: US
   6. Database server instance id (dbInstance).:  db2inst1
   7. Database server instance password (dbInstancePW).:  xxxxxxxx
   8. Database userid (dbUser).................:  db2inst1
   9. Database userid password (dbUserPW)......:  xxxxxxxx
  10. Path and file name of DDL file (ddlFile).:  /opt/IBM/TWSS/ServicePlatform/UsageDbDb2.ddl
  11. Database (re)create (dbCreate)...........:  false
      Recommended Value: TRUE
  12. Local Database (dbLocal) ................:  true
      Recommended Value: FALSE
  13. Remote Database Server Node Name (dbNodeName)...:  RDBSRV
      Recommended Value: RDBSRV
```

Figure B-17 Installation of the Third Party Call Control Database configuration

12. Execute the following command:

    ./crtsrvDb2.sh <fully qualified hostname> 50000 TWSS61 TWSS US db2inst1 xxxxxxxx db2inst1 xxxxxxxx /opt/IBM/TWSS/IMSServices/AdminTpcDb2.ddl false true

    Press Enter twice to create the database configuration.



```
[db2inst1@kofrzoy IMSServices]$ ./crtsrvDb2.sh kofrzoy.itso.ral.ibm.com 50000 TWSS61 TWSS US db2inst
1 xxxxxxxx db2inst1 xxxxxxxx /opt/IBM/TWSS/IMSServices/AdminTpcDb2.ddl false true
Usage: crtsrvDb2.sh dbServer dbPort dbName dbAlias dbLocale
                    dbInstance dbInstancePW dbUser dbUserPW
                    ddlFile dbCreate dbLocal dbNodeName menuflag
These values will be used to create the database.
Each value must be filled in.
The menuflag should be set to TRUE to display the menu.

Press Enter to accept these values and continue, otherwise
  enter the number of the parameter you wish to override.

   1. Database server hostname (dbServer)......:  kofrzoy.itso.ral.ibm.com
      Recommended Value: fully qualified host name.
   2. Database server connection port (dbPort).:  50000
   3. Database name (dbName)...................:  TWSS61
      Recommended Value: SOA610D
   4. Database alias (dbAlias).................:  TWSS
      Recommended Value: SOA610
   5. Database locale (dbLocale)...............:  US
      Recommended Value: US
   6. Database server instance id (dbInstance).:  db2inst1
   7. Database server instance password (dbInstancePW).:  xxxxxxxx
   8. Database userid (dbUser).................:  db2inst1
   9. Database userid password (dbUserPW)......:  xxxxxxxx
  10. Path and file name of DDL file (ddlFile).:  /opt/IBM/TWSS/IMSServices/AdminTpcDb2.ddl
  11. Database (re)create (dbCreate)...........:  false
      Recommended Value: TRUE
  12. Local Database (dbLocal) ................:  true
      Recommended Value: FALSE
  13. Remote Database Server Node Name (dbNodeName)...:  RDBSRV
      Recommended Value: RDBSRV
```

Figure B-18 Installation of the Third Party Admin Database configuration

B.2.4 Configure Service Integration Bus

TWSS utilizes the WebSphere Application Server Service Integration (SI) bus to provide the Java Message Service (JMS) messaging platform. The following steps are the instructions to setup the SI Bus for TWSS:

1. Login to the system as root.
2. Switch to the TWSS WebSphere Application Server profile directory:

   `cd /opt/IBM/WebSphere/AppServer/profiles/AppSrv05/bin`

   Where, AppSrv05 is the profile hosting TWSS.
3. Start the application server. Enter:

   `./startServer.sh server1`
4. Load a compatible Internet Browser (such as FireFox, Mozilla or Internet Explorer®).
5. Login to the Integrated Solutions Console.

d. Navigate to the Integrated Solutions Console for the appropriate application server, for example:

`http://<hostname>:9064/ibm/console`

e. Enter the appropriate username and password.

6. Create a Service Integration bus.

a. Click **Service integration** → **Buses** in the navigation panel.

b. Click **New**.

c. Enter `PXNotifyBus`.

d. Deselect Bus security.

e. Click **Next**.

f. Click **Finish**.

g. Click **Save** to save changes to the master configuration.

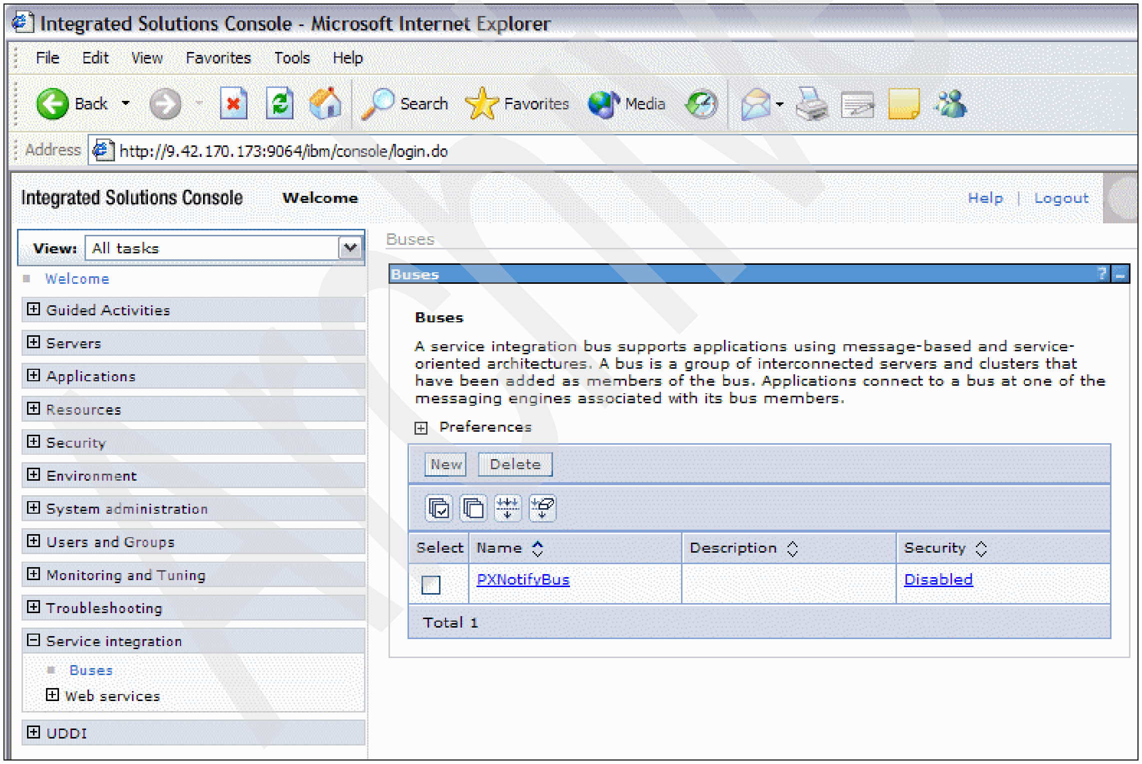


Figure B-19 Create PXNotifyBus SI bus

7. Add the Application Server as a member of the SI Bus.
   a. Click **PXNotifyBus**.
   b. Under Topology, click **Bus members**.
   c. Click **Add**.
   d. Click **Server**, then click **Next**.
   e. Click **File store**, then click **Next**.
   f. Click **Next**.
   g. Click **Finish**.
   h. Click **Save** to save changes to the master configuration.

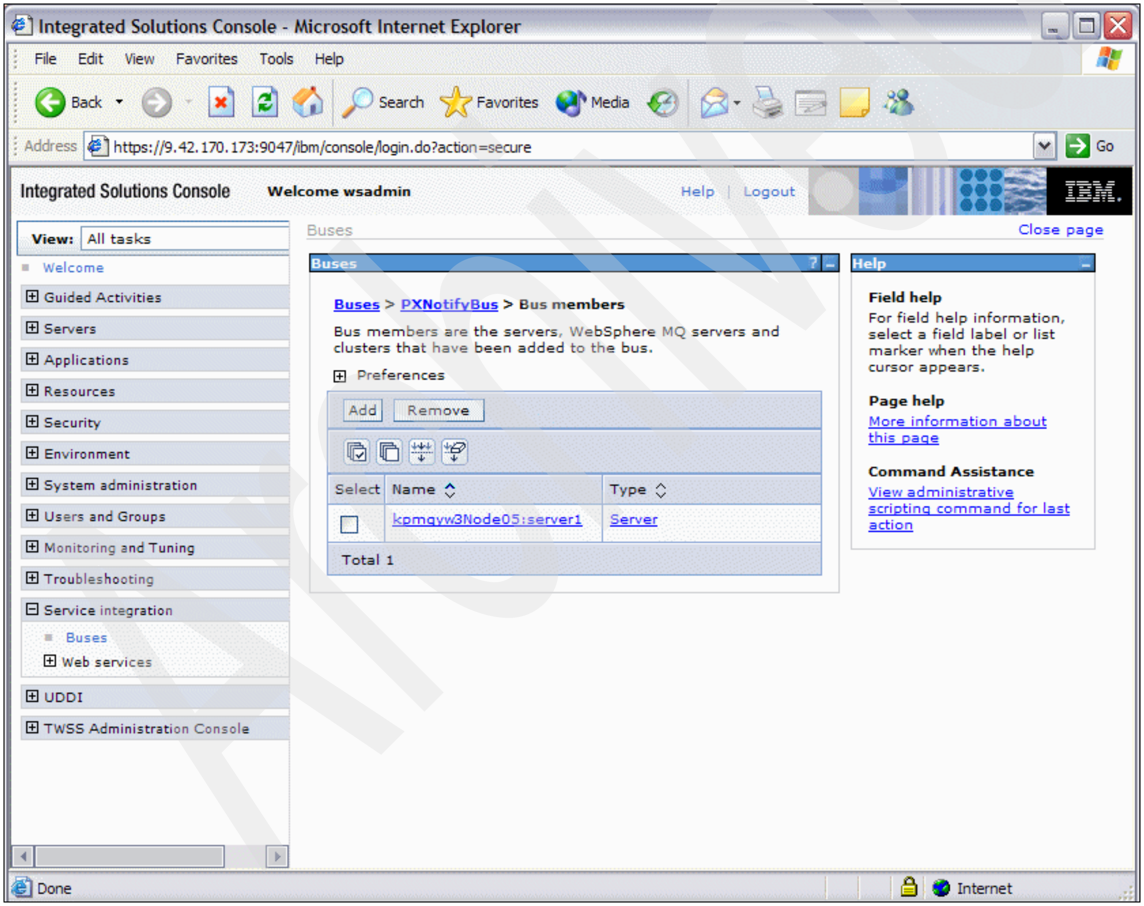


Figure B-20 Add an Application Server to the SI Bus

8. Add a bus destination.
   a. Click **PXNotifyBus**.
   b. Click **Destinations**.
   c. Click **New**.
   d. For destination type, select **Queue**. Click **Next**.
   e. In the Identifier field, enter: `PXNotifyDestination`. Click **Next**.
   f. Select the Bus Member.
   g. Click **Next**.
   h. Click **Finish**.
   i. Click **Save** to save changes to the master configuration.

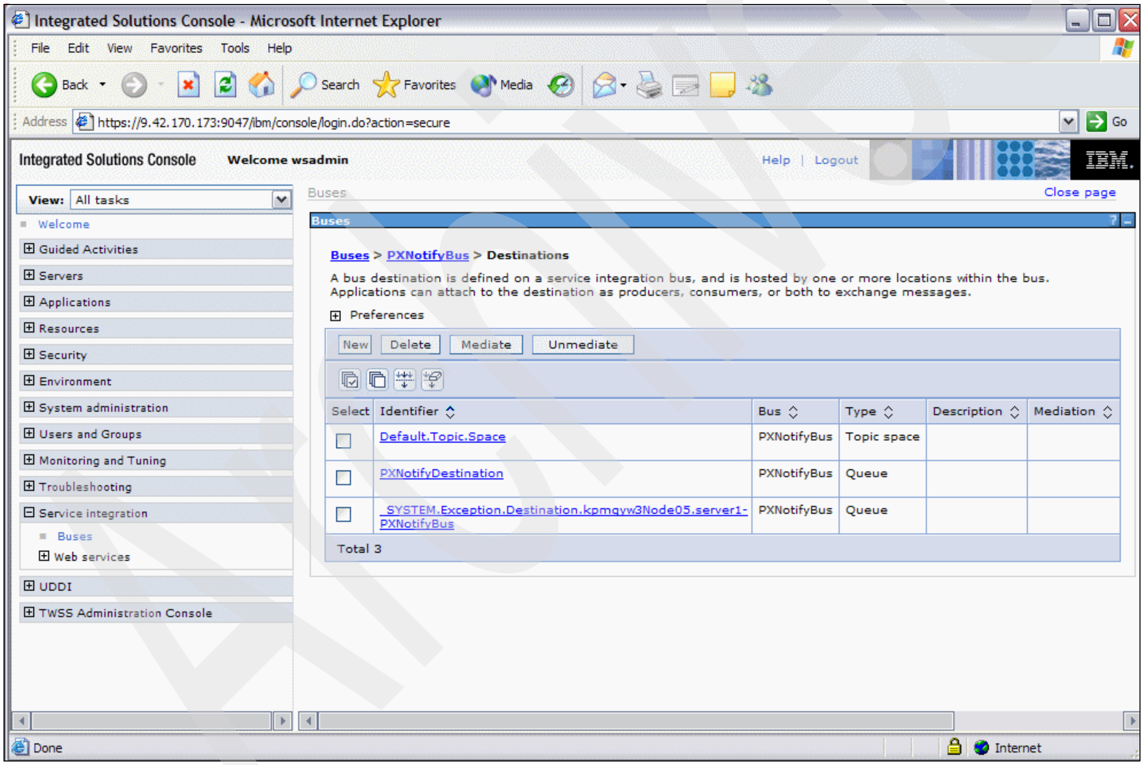


Figure B-21 Add a bus destination to the SI Bus

9. Create the JMS connection factory.
   a. In the navigation panel, click **Resources** → **JMS** → **JMS providers**.
   b. Verify Scope is set to Node; if not, expand **Scope**.

c. Select your node from the drop-down list.

d. Click **Default messaging provider**.

e. Under Additional Properties, click **Queue connection factories**.

f. Click **New**.

g. In the Name field, enter: `PXNotifyConnectionFactory`.

h. In the JNDI name field, enter: `jms/PXNotifyConnectionFactory`.

i. For the Bus name, select **`PXNotifyBus`**.

j. Click **Apply**.

k. Click **Save** to save changes to the master configuration.

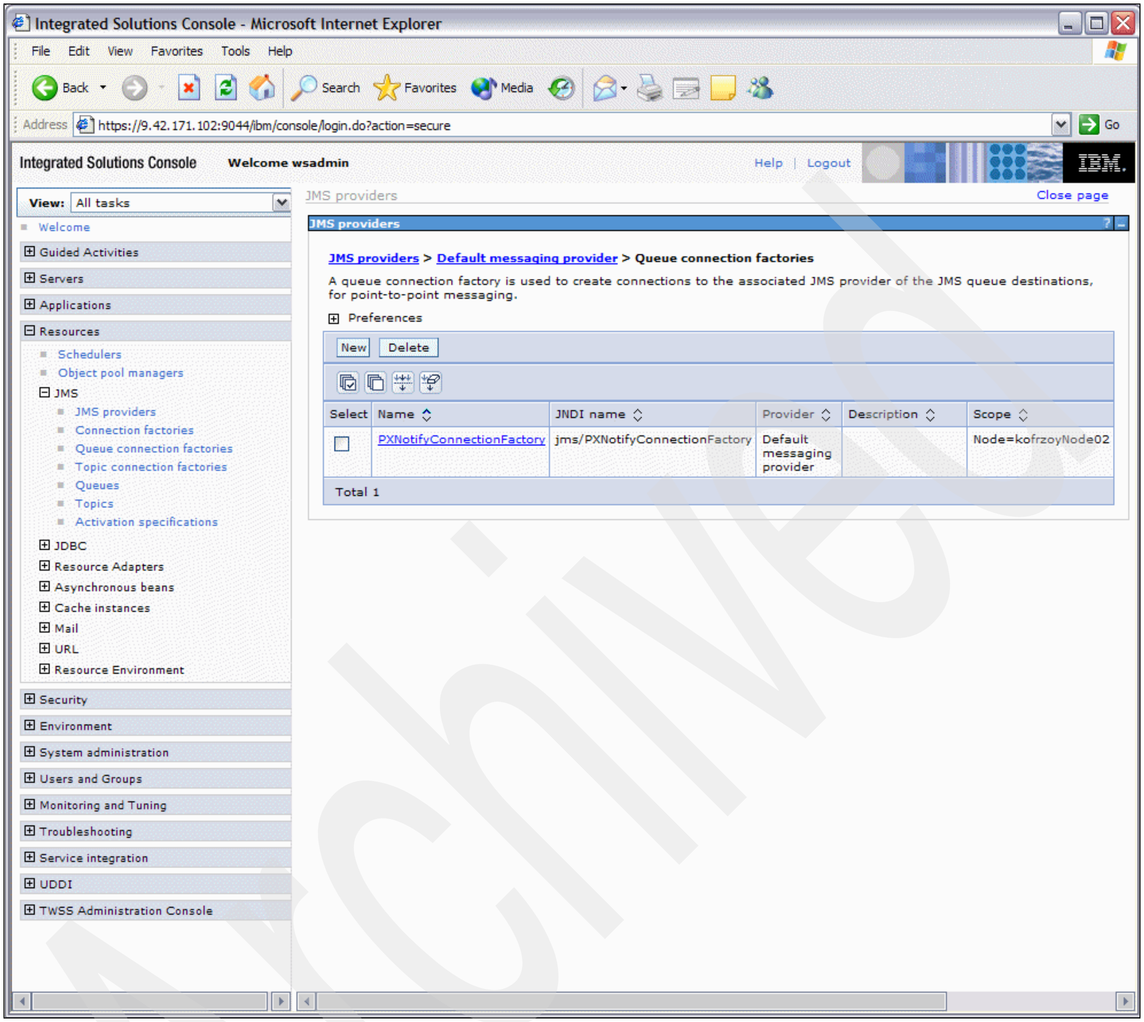


Figure B-22 Defining the Connection Factory

10. Create the JMS queue.

    a. In the navigation panel, click **Resources** → **JMS** → **JMS providers**.

    b. Click **Default messaging provider**.

    c. Under Additional Properties, click **Queues**.

    d. Click **New**.

    e. In the Name field, enter: `PXNotifyQueue`.

    f. In the JNDI name field, enter: `jms/PXNotifyQueue`.

g. For the Bus name, select **PXNotifyBus**.

h. For Queue name, select **PXNotifyDestination**.

i. Click **Apply**.

j. Click **Save** to save changes to the master configuration.

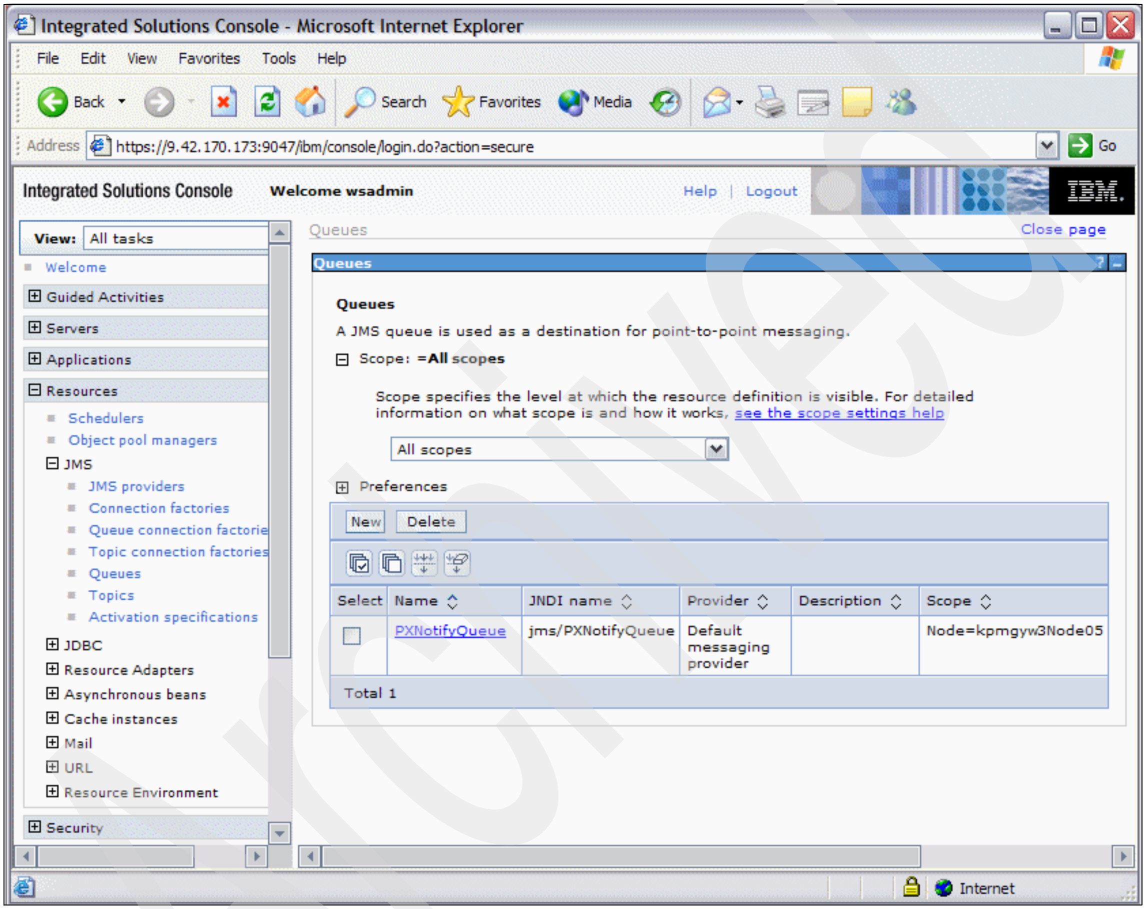


Figure B-23 Define the JMS Queue

11.Create the JMS activation specification.

a. In the navigation panel, click **Resources** → **JMS** → **JMS providers**.

b. Click **Default messaging provider**.

c. Under Additional Properties, click **Activation specifications**.

d. Click **New.**

e. In the Name field, enter: `PX Notification Activation Spec`

f. In the JNDI Name field, enter: `eis/PXNotifyActivationSpec`

g. For Destination type, select **queue**.

h. In the Destination JNDI Name field, enter `jms/PXNotifyQueue`

i. For the Bus name, select **PXNotifyBus**.

j. Click **Apply**.

k. Click **Save** to save changes to the master configuration.

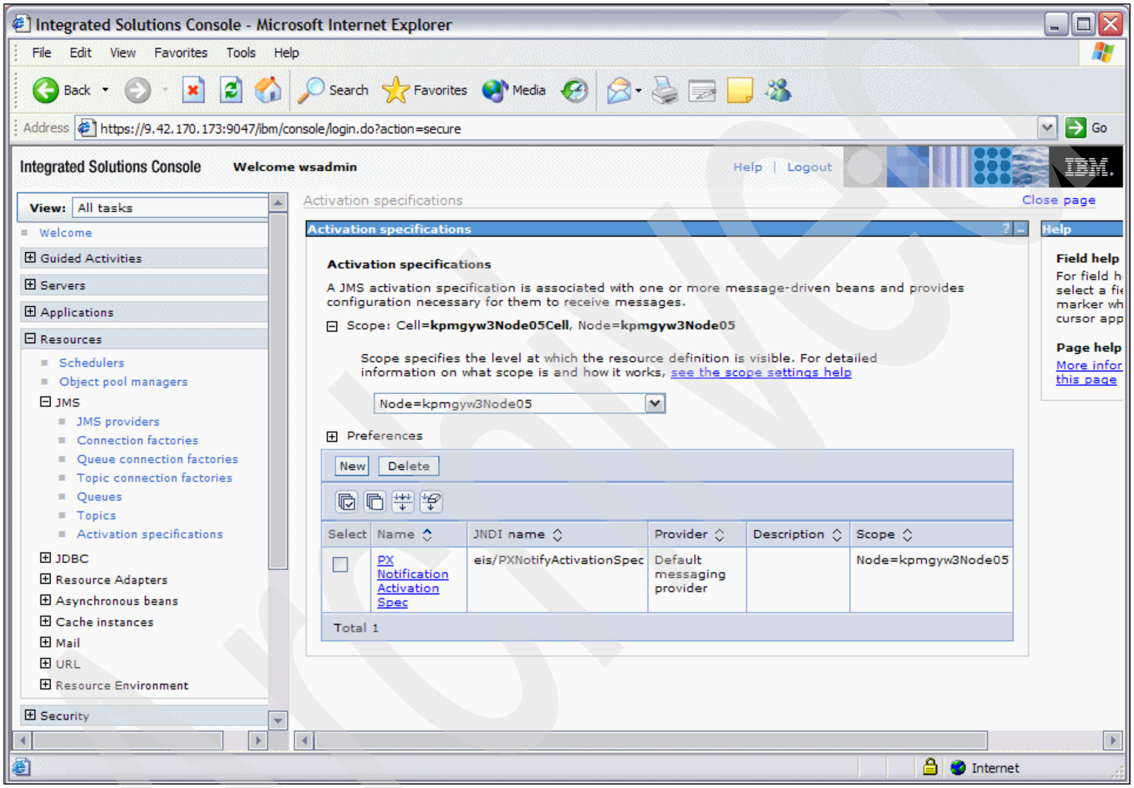


Figure B-24 Define Activation Specification for JMS Provider

B.2.5 Configure JDBC

The WebSphere Application Server communicates with the DB2 database configured in B.2.3, “Configure DB2” on page 546. To configure the communication, perform these tasks:

- Create authentication alias on page 562
- Create JDBC Provider on page 563
- Create the data source on page 565

Create authentication alias

1. In the navigation panel, click **Security** → **Secure administration, applications, and infrastructure**.
2. Expand **Java Authentication and Authorization Service**.
3. Click **J2C authentication data**.
4. Click **New**.
5. In the Alias field, enter: `db2 local alias`
6. In the User ID, enter: `db2inst1`

   This is the user_ID that will be used to access the TWSS database.
7. In the Password field, enter the password for db2inst1 user_ID that you entered above.
8. Click **OK**.
9. Click **Save** to save the changes to the master configuration.

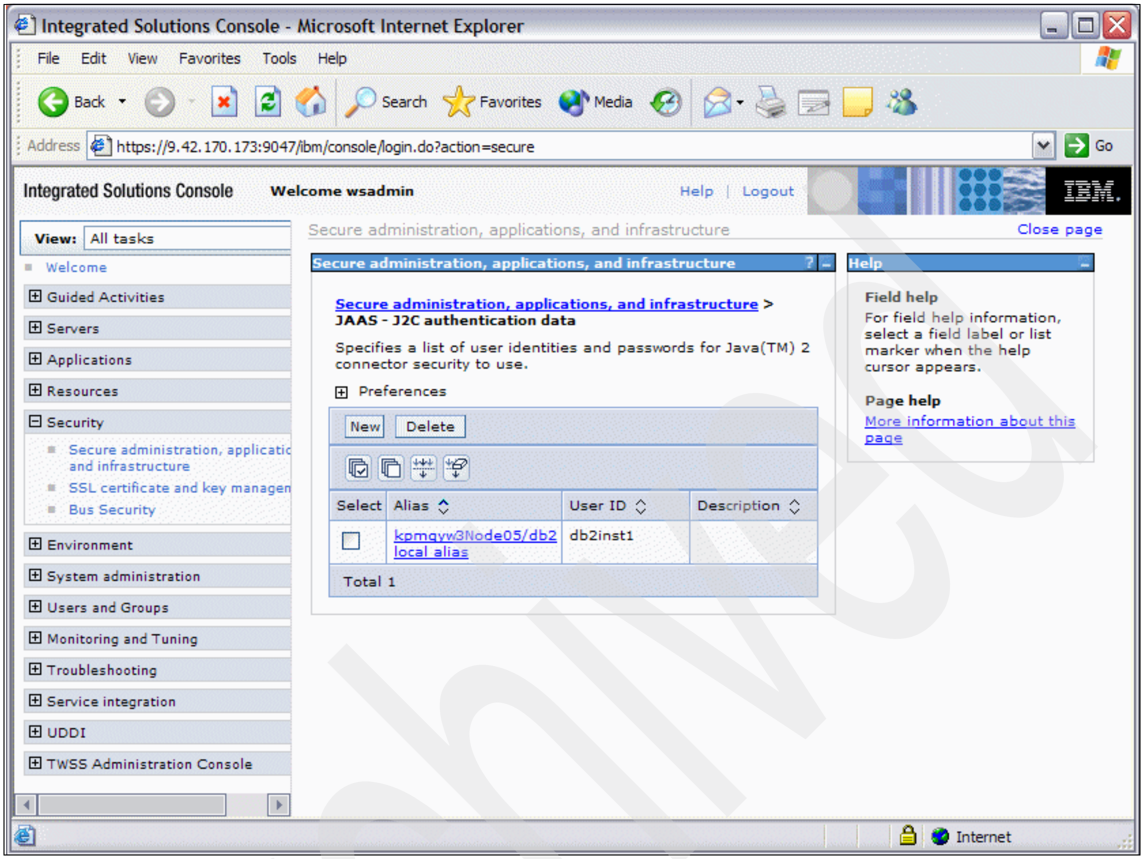


Figure B-25 J2C Authentication Alias defined for JDBC datasource

Create the JDBC Provider

1. In the navigation panel, click **Resources** → **JDBC** → **JDBC Providers**.
2. Expand **Scope**.
3. Select your node from the drop-down list.
4. Click **New**.
5. Select **DB2** as the database type from the drop-down list.
6. Select **DB2 Universal JDBC Driver Provider** for the Provider type.
7. Select **Connection pool data source** for the Implementation type.
8. Click **Next**.

9. Enter the db2 class paths for your installation; these are normally:

```
/home/db2inst1/sqllib/java
/home/db2inst1/sqllib/lib
```

10. Click **Next**.

11. Verify all values are correct.

12. Click **Finish**.

13. Click **Save** to save the changes to the master configuration.

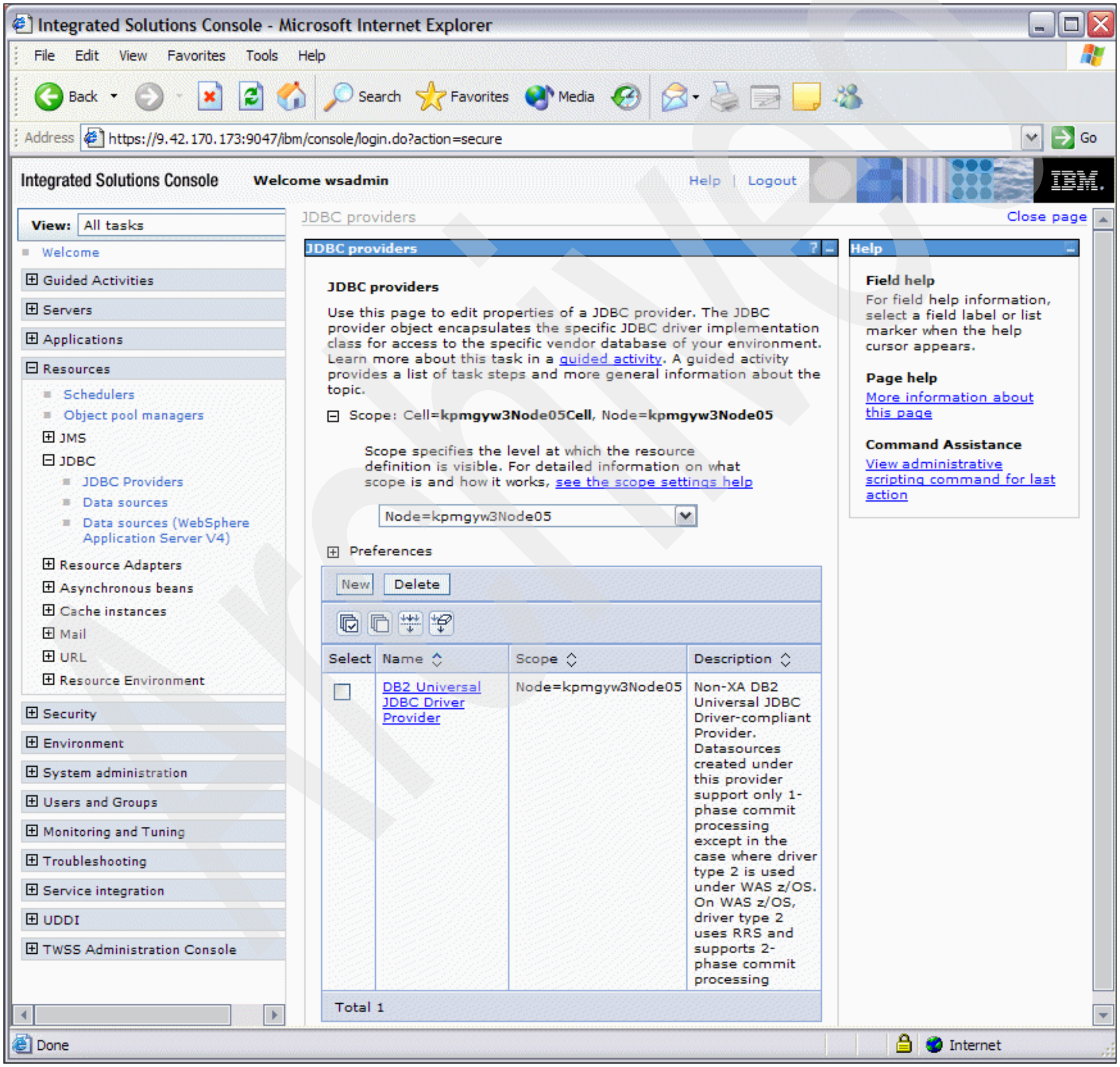


Figure B-26 JDBC Provider defined

Create the data source

1. Click the JDBC provider, created in “Create the JDBC Provider” on page 563.
2. Click **Data sources**.
3. Click **New**.
4. In the JNDI name field, enter: `jdbc/SOADB`
5. From the Component-managed authentication alias drop-down list, select **db2 local alias**.
6. Click **Next**.
7. In the Database name field, enter **TWSS**.
8. In the Driver type field, select **Type 4** to specify the connectivity type of the data source.
9. In the Server name field, enter the <server_name> (`localhost`).
10. In the Port number field, enter the port_number (normally `50000`).
11. Click **Next**.
12. Verify the values are correct.
13. Click **Finish**.
14. Click **Save** to save the changes to the master configuration.

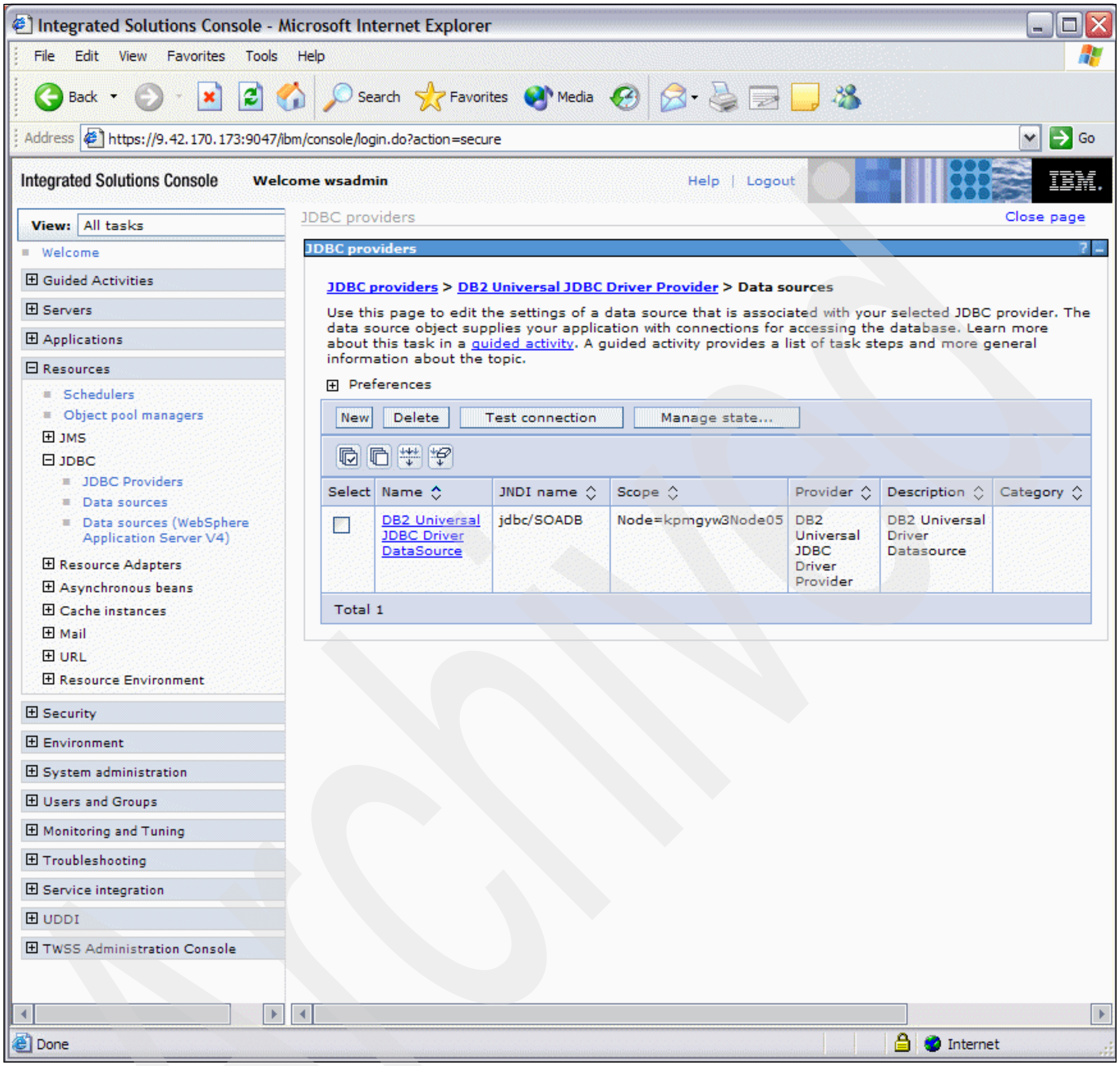


Figure B-27 Data Source created for TWSS

15. To test the database connection:

   a. Select the tick box of the created data source.

   b. Click **Test Connection**.

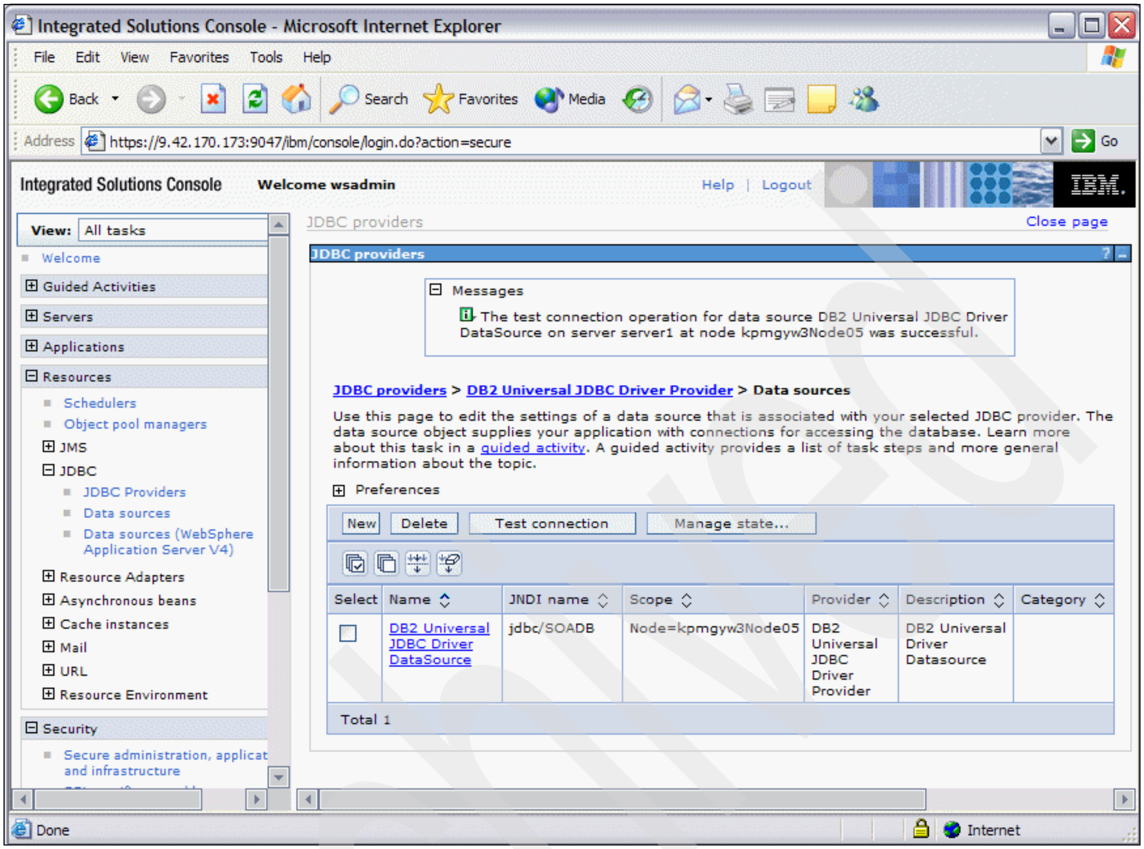


Figure B-28   Data source tested successfully

B.2.6 Tune the Application Server

Tuning of the Application Server depends on the services required by TWSS. The following steps describe the process for tuning the application server for Third Party Call Control For high performance Web service connections, enable in process connections (enableInProcessConnections).

> **Note:** You can find additional instructions for tuning the application server in the TWSS Information Center at:
>
> http://publib.boulder.ibm.com/infocenter/wtelecom/v6r1/index.jsp?topic=/com.ibm.twss.javadoc.doc/adminconpublicjavadoc/com/ibm/soa/common/mbean/package-summary.html

1. In the Integrated Solutions Console navigation panel, click **Servers** → **Application Servers**.
2. Select the application server where the services are deployed.
3. Expand **Web Container Settings**.
4. Click **Web Container**.
5. Click **Custom Properties**.
6. Under Additional Properties.
    a. Click **New**.
    b. For the Name, enter: `enableInProcessConnections`
    c. For the Value, enter: `true`
    d. Click **OK**.
7. Click **Save**.

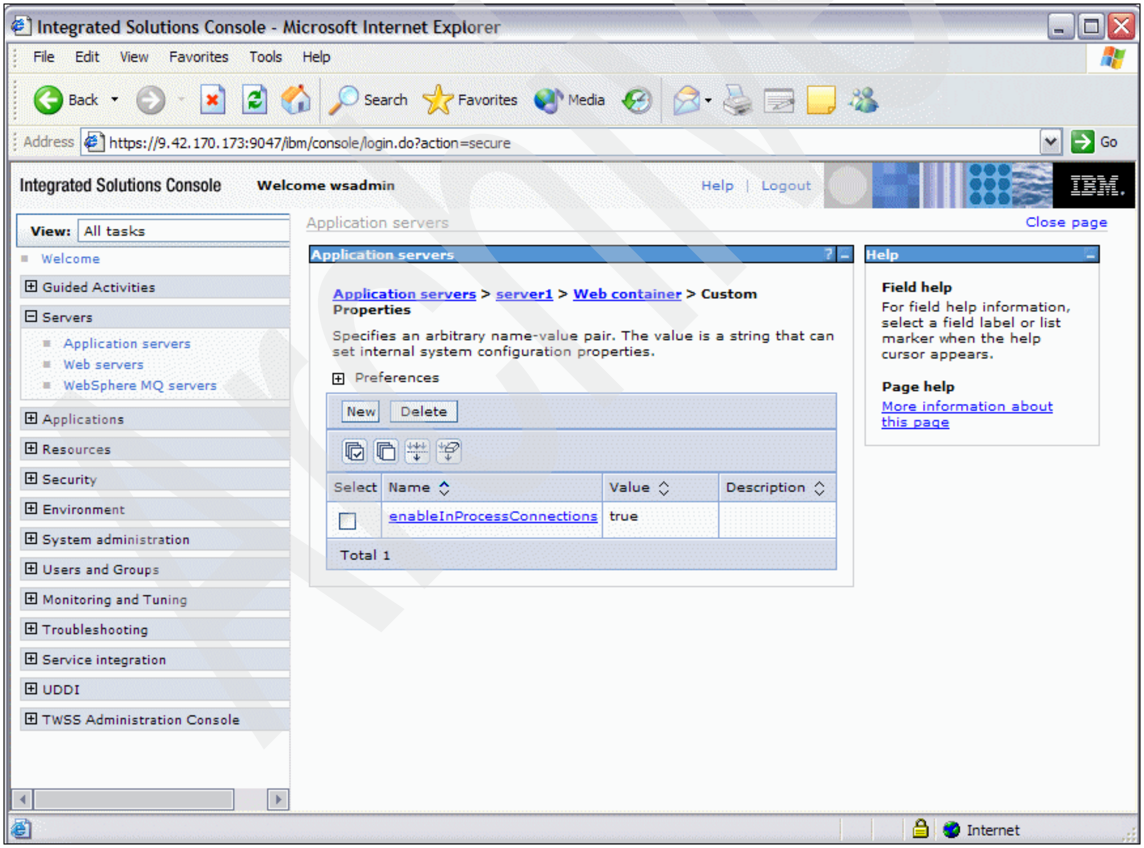


Figure B-29 Web Container configuration for TWSS

Enable the EJB Container data source.

1. In the Integrated Solutions Console navigation panel, click **Servers** → **Application Servers**.
2. Select the application server where the services are deployed.
3. Under Container Settings:
   a. Click **EJB Container Settings**.
   b. Click **EJB container**.
4. From the Default data source JNDI name menu:
   a. Select **jdbc/SOADB**.
   b. Click **OK**.
5. Click **Save**.

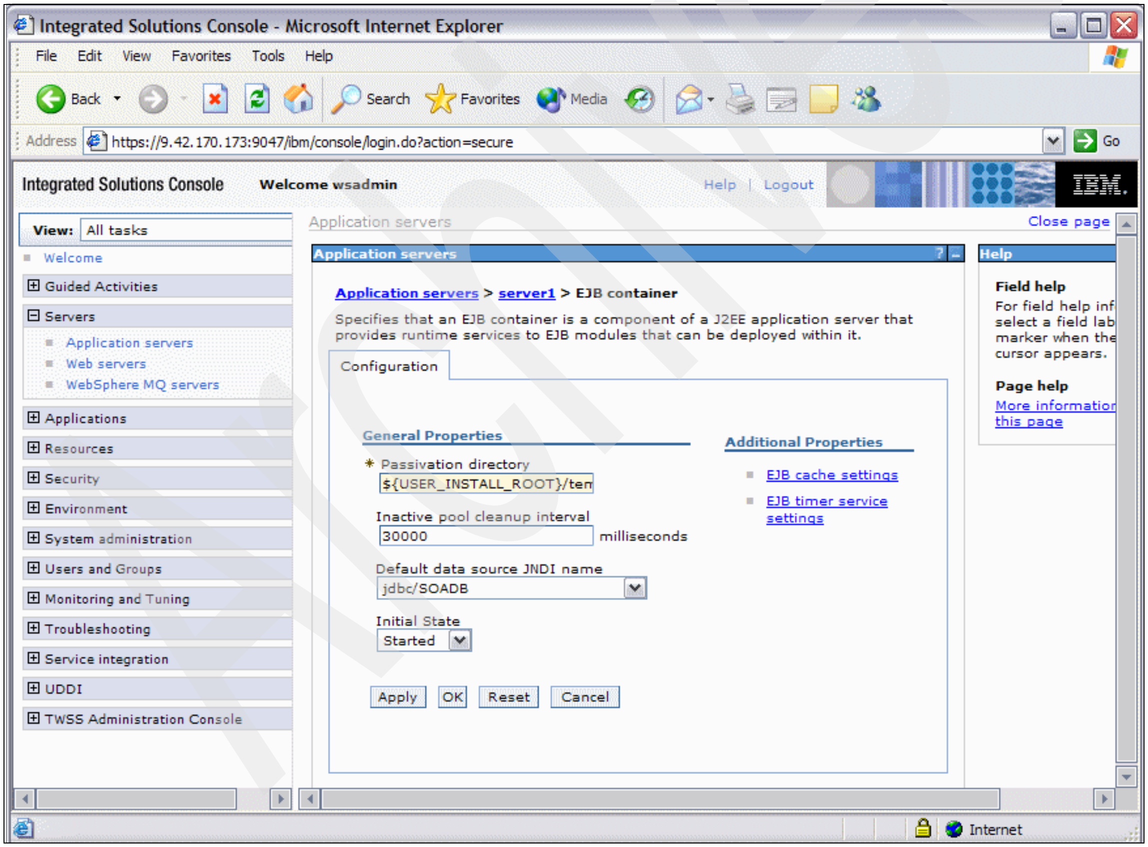


Figure B-30 EJB Container Configuration for TWSS

B.2.7 Deploy TWSS Applications

TWSS is divided into a number of EAR files for installation. The number of files you have to install depends on the required functionality. There are base EAR files that must be installed, and then there are application specific EAR files. The following process describes the installation steps for the base EAR files, and for Third Party Call Control functionality.

> **Important:** You must ensure that the application server is restarted after the creation of the SI Bus and JMS configuration, some application will fail to start successfully if you do not restart the application server.

The topics in this section are:

- "Install the admission EAR file" on page 570
- "Install the fault and alarm EAR file" on page 572
- "Install the network resources EAR file" on page 573
- "Install the PX notification EAR file" on page 575
- "Install the traffic shaping EAR file" on page 577
- "Install the usage EAR file" on page 578
- "Install the Third Party Call Web Service" on page 580
- "Restart the application server" on page 581

Install the admission EAR file

1. In the Integrated Solutions Console navigation panel, click **Applications** → **Enterprise Applications**.
2. Click **Install**.
3. Click **Browse** to locate `admission.ear`.
4. Select **Show me all installation options and parameters**.
5. Click **Next**.
6. On the Preparing for the application installation page, retain the defaults. Click **Next**.
7. On the Application Security Warnings page, click **Continue**.
8. On the Select installation options page, retain the defaults. Click **Next**.
9. On the Map modules to servers page, select the check box for each module. Click **Next**.
10. On the Select current backend ID page, select the correct database type: `DB2UDBNT_V8_1` for DB2

    Click **Next**.

11. On the Provide JSP reloading options for Web modules page, retain the defaults. Click **Next**.
12. Click **Map default data sources for modules containing 2.x entity beans**. Select the check box EJB modules.
13. In the Specify authentication method area, select the **db2 local alias** for the Authentication data entry. Click **Apply.**
14. Verify that the correct JNDI name is selected: `jdbc/SOADB`
15. Click **Next**.
16. Click the **Map security roles to users or groups** link.
17. Map the SOAAdministrator to the desired users, groups.
    a. Select the check box for the SOAAdministrator.
    b. Click **Look up users**.
    c. Click **Search**.
    d. Select **wsadmin**.
    e. Click the Add button >>.
    f. Click **OK**.
18. Map the AdmissionControlRole to the desired users, groups, or both.
    a. Select the check box for the AdmissionControlRole.
    b. Click **Look up users**.
    c. Click **Search**.
    d. Select **wsadmin**.
    e. Click the Add button **>>.**
    f. Click **OK**.
19. Click **Next**.
20. On the Ensure all unprotected 2.x methods have the correct level of protection page. Verify that the Uncheck option is selected.
21. Click **Next**.
22. Review the summary information.
23. Click **Finish**.
24. Wait while the EAR is deployed.
25. Click **Save**.
26. Wait while the EAR is published.
27. Start the application to verify it deployed correctly.

a. Select the check box for the application.

b. Click **Start**.

Install the fault and alarm EAR file

1. In the Integrated Solutions Console navigation panel, click **Applications** → **Enterprise Applications**.
2. Click **Install**.
3. Click **Browse** to locate the `faultalarm.ear` file.
4. Select **Show me all installation options and parameters**.
5. Click **Next**.
6. On the Preparing for the application installation page, retain the defaults. Click **Next**.
7. On the Application Security Warnings page, click **Continue**.
8. On the Select installation options page, retain the defaults. Click **Next**.
9. On the Map modules to servers page, select the check box for each module. Click **Next**.
10. On the Select current backend ID page, select the correct database type: `DB2UDBNT_V8_1` for DB2

    Click **Next**.
11. Select the **Map default data sources for modules containing 2.x entity beans** link.
12. Select the check box EJB modules.
    a. In the Specify authentication method area, select the **db2 local alias** for the Authentication data entry. Click **Apply**.
    b. Verify that the correct JNDI name is selected, `jdbc/SOADB`
    c. Click **Next**.
13. Select the **Map security roles to users or groups** link.
14. Map the roles to users or groups.
    a. Select the check box for the SOAAdministrator.
    b. Click **Look up users**.
    c. Click **Search**.
    d. Select **wsadmin**.
    e. Click the Add button **>>.**
    f. Click **OK**.

15. Map the FaultAlarmRole to the desired users, groups, or both.
    a. Select the check box for the FaultAlarmRole.
    b. Click **Look up users**.
    c. Click **Search**.
    d. Select **wsadmin**.
    e. Click the Add button **>>.**
    f. Click **OK**.
16. Click **Next**.
17. On the Ensure all unprotected 2.x methods have the correct level of protection page, verify that the Uncheck option is selected.
18. Click **Next**.
19. Review the summary information.
20. Click **Finish**.
21. Wait while the EAR is deployed.
22. Click **Save**.
23. Wait while the EAR is published.
24. Start the application to verify it deployed correctly.
    a. Select the check box for the application.
    b. Click **Start**.

Install the network resources EAR file

1. In the Integrated Solutions Console navigation panel, click **Applications** → **Enterprise Applications**.
2. Click **Install**.
3. Click Browse to locate the file `networkresources.ear`.
4. Select **Show me all installation options and parameters**.
5. Click **Next**.
6. On the Preparing for the application installation page, retain the defaults. Click **Next**.
7. On the Application Security Warnings page, click **Continue**.
8. Click **Select current backend ID**.
9. Select the correct database type `DB2UDBNT_V8_1` for DB2.
10. Click **Next**.

11. Click the **Map default data sources for modules containing 2.x entity beans** link.
12. Select the check box EJB modules.
13. In the Specify authentication method area, select **db2 local alias** for the Authentication data entry. Click **Apply**.
14. Verify that the correct JNDI name is selected: `jdbc/SOADB`

    Click **Next**.
15. Click **Map security roles to users or groups**.
16. Map the SOAAdministrator roles to users or groups.
    a. Select the check box for the SOAAdministrator.
    b. Click **Look up users**.
    c. Click **Search**.
    d. Select **wsadmin**.
    e. Click the Add button **>>**.
    f. Click **OK**.
17. Map the NetworkResourcesRole to the desired users, groups, or both.
    a. Select the check box for the NetworkResourcesRole and either **Look up users**.
    b. Click **Search**.
    c. Select **wsadmin**.
    d. Click the Add button **>>**.
    e. Click **OK**.
18. Click **Next**.
19. On the Ensure all unprotected 2.x methods have the correct level of protection page, verify that the Uncheck option is selected.
20. Click **Next**.
21. Review the summary information.
22. Click **Finish**.
23. Wait while the EAR is deployed.
24. Click **Save.**
25. Wait while the EAR is published.
26. Start the application to verify it deployed correctly.
    a. Select the check box for the application.

b. Click **Start.**

Install the PX notification EAR file

1. In the Integrated Solutions Console navigation panel, click **Applications** → **Enterprise Applications.**
2. Click **Install.**
3. Click Browse to locate the `pxnotification.ear` file.
4. Select **Show me all installation options and parameters.**
5. Click **Next.**
6. On the Preparing for the application installation page, retain the defaults. Click **Next.**
7. On the Application Security Warnings page, click **Continue.**
8. On the Select installation options page, retain the defaults. Click **Next.**
9. On the Map modules to servers page, select the check box for each module. Click **Next.**
10. On the Select current backend ID page, select the correct database type: `DB2UDBNT_V8_1` for DB2.

    Click **Next.**
11. Click **Bind listeners for message-driven beans.**
    a. Select the check box for pxnotify-delivery-ejb EJB module.
    b. Click **Apply Multiple Mappings.**
    c. In the Activation Specification region, select **db2 local alias** as the AuthenticationSpec authentication alias. Click **Apply.**
    d. Click **Next.**
12. Click **Map default data sources for modules containing 2.x entity beans.**
13. Select the check box EJB modules.
14. In the Specify authentication method area, select the **db2 local alias** for the Authentication data entry. Click **Apply.**
15. Verify that the correct JNDI name is selected: `jdbc/SOADB`

    Click **Next.**
16. On the Map data sources for all 2.x CMP beans page, retain the defaults. Click **Next.**
17. On the Map resource references to resources page.
    a. Select the radio button for Use default method (many-to-one mapping).

b. From the Authentication data entry drop-down list, select **db2 local alias.**

c. Select the pxnotify-delivery-web check box.

d. Click **Apply.**

e. Select the check box for the pxnotify-delivery-web module. Click **Browse.**

f. Select the PXNotifyConnectionFactory check box. Click **Apply.**

g. For the javax.jms.Queue:

i. Click **Browse.**

ii. Select the PXNotifyQueue check box.

iii. Click **Apply.**

h. Click **Next.**

18. Click the **Map security roles to users or groups link.**

19. Map the SOAAdministrator to users or groups.

a. Select the check box for the SOAAdministrator.

b. Click **Look up users.**

c. Click **Search.**

d. Select **wsadmin.**

e. Click the Add button **>>.**

f. Click **OK.**

20. Map the PxNotifyRole to the desired users, groups, or both.

a. Select the check box for the PxNotifyRole.

b. Click **Look up users.**

c. Click **Search.**

d. Select **wsadmin.**

e. Click the Add button **>>.**

f. Click **OK.**

21. Click **Next.**

22. On the Ensure all unprotected 2.x methods have the correct level of protection page, verify that the Uncheck option is selected.

23. Click **Next.**

24. Review the summary information.

25. Click **Finish.**

26. Wait while the EAR is deployed.

27. Click **Save.**
28. Wait while the EAR is published.
29. Start the application to verify it deployed correctly.
    a. Select the check box for the application.
    b. Click **Start.**

Install the traffic shaping EAR file

1. In the Integrated Solutions Console navigation panel, click **Applications → Enterprise Applications.**
2. Click **Install.**
3. Click **Browse** to locate the `traffic.ear` file on your local system.
4. Select **Show me all installation options and parameters.**
5. Click **Next.**
6. On the Preparing for the application installation page, retain the defaults. Click **Next.**
7. On the Application Security Warnings page, click **Continue.**
8. On the Select installation options page, retain the defaults. Click **Next.**
9. On the Map modules to servers page, select the check box for each module. Click **Next.**
10. On the Select current backend ID page, select the correct database type: `DB2UDBNT_V8_1` for DB2.

    Click **Next.**
11. Click the **Map default data sources for modules containing 2.x entity beans** link.
12. Select the check box EJB modules.
13. In the Specify authentication method area, select the **db2 local alias** for the Authentication data entry. Click **Apply.**
14. Verify that the correct JNDI name is selected: `jdbc/SOADB`

    Click **Next.**
15. Click the **Map security roles to users or groups** link.
16. Map the SOAAdministrator to users or groups.
    a. Select the check box for the SOAAdministrator.
    b. Click **Look up users.**
    c. Click **Search.**

   d. Select **wsadmin.**

   e. Click the Add button **>>.**

   f. Click **OK.**

17. Map the TrafficShapingRole to the desired users, groups, or both.

   a. Select the check box for the TrafficShapingRole.

   b. Click **Look up users.**

   c. Click **Search.**

   d. Select **wsadmin.**

   e. Click the Add button **>>.**

   f. Click **OK.**

18. Click **Next.**

19. On the Ensure all unprotected 2.x methods have the correct level of protection page, verify that the Uncheck option is selected.

20. Click **Next.**

21. Review the summary information.

22. Click **Finish.**

23. Wait while the EAR is deployed.

24. Click **Save.**

25. Wait while the EAR is published.

26. Start the application to verify it deployed correctly.

   a. Select the check box for the application.

   b. Click **Start.**

Install the usage EAR file

1. In the Integrated Solutions Console navigation panel, click **Applications** → **Enterprise Applications.**

2. Click **Install.**

3. Click **Browse** to locate the `usage.ear` file on your local system.

4. Select **Show me all installation options and parameters.**

5. Click **Next.**

6. On the Preparing for the application installation page, retain the defaults, Click **Next.**

7. On the Application Security Warnings page, click **Continue.**

8. On the Select installation options page, retain the defaults. Click **Next.**
9. On the Map modules to servers page, select the check box for each module. Click **Next.**
10. On the Select current backend ID page, select the correct database type: `DB2UDBNT_V8_1` for DB2

    Click **Next.**
11. Click the **Map default data sources for modules containing 2.x entity beans** link.
12. Select the check box EJB modules.
13. In the Specify authentication method area, select the **db2 local alias** for the Authentication data entry. Click **Apply.**
14. Verify that the correct JNDI name is selected: `jdbc/SOADB`

    Click **Next.**
15. Click the **Map security roles to users or groups** link.
16. Map the SOAAdministrator to users or groups..
    a. Select the check box for the SOAAdministrator.
    b. Click **Look up users.**
    c. Click **Search.**
    d. Select **wsadmin.**
    e. Click the Add button >>.
    f. Click **OK.**
17. Map the UsageRecordRole to the desired users, groups, or both.
    a. Select the check box for UsageRecordRole.
    b. Click **Look up users.**
    c. Click Search.
    d. Select **wsadmin.**
    e. Click the Add button >>.
    f. Click **OK.**
18. Click **Next.**
19. On the Ensure all unprotected 2.x methods have the correct level of protection page, verify that the Uncheck option is selected.
20. Click **Next.**
21. Review the summary information.

22. Click **Finish.**
23. Wait while the EAR is deployed.
24. Click **Save.**
25. Wait while the EAR is published.
26. Start the application to verify it deployed correctly.
    a. Select the check box for the application.
    b. Click **Start.**

Install the Third Party Call Web Service

1. In the Integrated Solutions Console navigation panel, click **Applications → Enterprise Applications.**
2. Click **Install**.
3. Click **Browse** to locate the `thirdparty.ear` file on your local system.
4. Select **Show me all installation options and parameters.**
5. Click **Next.**
6. On the Preparing for the application installation page, retain the defaults. Click **Next.**
7. On the Application Security Warnings page, Click **Continue.**
8. On the Select installation options page, retain the defaults. Click **Next.**
9. On the Map modules to servers page, select the check box for each module. Click **Next.**
10. On the Select current backend ID page, select the correct database type: `DB2UDBNT_V8_1` for DB2,

    Click **Next.**
11. Click the **Map default data sources for modules containing 2.x entity beans** link.
12. Select the check box EJB modules.
13. In the Specify authentication method area:
    a. Select the **db2 local alias** for the Authentication data entry.
    b. Click **Apply.**
    c. Verify that the correct JNDI names are selected: `jdbc/SOADB`
    d. Click **Next.**
14. Click the **Map security roles to users or groups** link.
15. Map the SOAAdministrator to users or groups.

   a. Select the check box for the SOAAdministrator.
   b. Click **Look up users.**
   c. Click **Search.**
   d. Select **wsadmin.**
   e. Click the Add button >>.
   f. Click **OK.**
16. Map the ThirdPartyCall_IMS_Role to the desired users, groups, or both.
   a. Select the check box for the ThirdPartyCall_IMS_Role.
   b. Click **Look up users.**
   c. Click **Search.**
   d. Select **wsadmin.**
   e. Click the Add button >>.
   f. Click **OK.**
17. Click **Next.**
18. On the Ensure all unprotected 2.x methods have the correct level of protection page, verify that the Uncheck option is selected.

    Click **Next.**
19. Review the summary information.
20. Click **Finish.**
21. Wait while the EAR is deployed.
22. Click **Save.**
23. Wait while the EAR is published.
24. Start the application to verify it deployed correctly.
   a. Select the check box for the application.
   b. Click **Start.**

Restart the application server

Once all the applications have been installed, you must restart the application to ensure the installation is successful.

1. Change directories.

   Enter: **`cd <was_profile_base>/bin`**
2. Stop the application server.

   Enter: **`./stopServer.sh server1 -user <username> -password <password>`**

3. Start the application server.

   Enter: `./startServer.sh server1`

B.2.8 Verify the installation

We verify the installation of the Third Party Call Control Web service by downloading the WSDL and testing it in the Application Server Toolkit. The process is outlined in the following steps:

1. Log into the Integrated Solutions Console and select **Applications** → **Enterprise Applications.**
2. Select **IMS Third PartyCall** application.
3. Under the Web Service Properties.
   a. Select **Publish WSDL files.**
4. Select the **IMS Third Party Call_WSDLFiles.zip.**
5. Save the file onto the local file system.
6. Unzip the downloaded file onto the local file system.
7. Start the Application Server Toolkit.
   a. Create a new Dynamic Web Project.
      i. Select **File** → **New** → **Project.**
      ii. Select **Web** → **Dynamic Web Project**.
      iii. Click **Next.**
   b. In the Project Name, enter: `3rdPartyWebProject`
   c. Click **Finish.**
8. Right-click the project.
9. Select **Import.**
10. Select **File System.**
11. Click **Next.**
12. Enter the location of the unzipped WSDL in the **From Directory.**
13. Select the WSDL files.
14. Click **Finish.**
15. Locate `px_tpc_s_2_1.wsdl` in the project.
    a. Right-click.
    b. Select **Web Services** → **Test with Web Services Explorer.**
16. Click **makeCall** in the Web Services Explorer.

17. Fill in two valid SIP addresses.
18. Click **Go.**

This will setup a call below the two SIP endpoints specified.

B.2.9 Troubleshoot the installation

If the verification test failed, start your troubleshooting by reviewing the SOAP response and the SystemOut.log installed at <was_profile_base>/logs/server1. The following are common issues that you might run into, and the respective resolutions.

The topics in this section are:

- "WSDL invocation failure" on page 583
- "Unable to communicate with the core TWSS Web Services" on page 583
- "No admission control limits for the operation makeCall" on page 585
- "Network Resource Retrieval" on page 585

WSDL invocation failure

The WSDL invocation failed with the error message of:

`com.ibm.soa.parlayx21.common.ServiceException: SVC0001: A service error occurred. Error code is 0001.` The SystemOut.log file has the following error "`Error 404: No target servlet configured for uri: /soa/service_platform/privacy/services/PrivacyInterface`"

To resolve the problem follow these steps:

1. Log into the Integrated Solutions Console.
2. Select **TWSS Administration Console** → **Web Services** → **Single Servers.**
3. Select **IMS Third Party Call.ear.**
4. Select **Privacy Client.**
5. Delete the End Point URI value.
6. Click **OK.**
7. Click **Review.**
8. Click **Save.**

Unable to communicate with the core TWSS Web Services

The likely causes of this problem include the installation and initialization problems with TWSS application, and HTTP port number configuration (not being set to 9080).

Example 13-16 TWSS Core Web Services are not available

```
faultCode: HTTP
 faultString: ( 404 ) Not Found
 faultActor: http://localhost:9080
 faultDetail:
        null: WSWS3192E: Error: return code:  ( 404 ) Not Found
Error 404: No target servlet configured for uri:
/soa/service_platform/usage_record/services/UsageRecordInterface
```

To resolve the problem follow these steps:

1. Log into the Integrated Solutions Console.
2. Select **TWSS Administration Console** → **Web Services** → **Single Servers.**
3. Select **IMS Third Party Call.ear.**
4. Select **Admission Control Client.**
5. Modify the URI so it points to the correct port number for example, `http://localhost:`**`9084`**`/soa/service_platform/admission_control/services/AdmissionControlInterface`
6. Click **OK.**
7. Click **Review.**
8. Click **Save.**

Repeat Steps 4 to 8 for the following:

- Fault Alarm Client
- Traffic Shaping Client
- UsageRecord Client

9. Select **Web Services Platform** → **Single Servers.**
10. Select **Traffic Shaping.ear.**
11. Select **Network Resource Client.**
12. Modify the URI so it points to the correct port number for example, `http://localhost:`**`9084`**`/soa/service_platform/admission_control/services/AdmissionControlInterface`
13. Click **OK.**
14. Click **Review.**
15. Click **Save.**
16. Restart WebSphere Application Server.

No admission control limits for the operation makeCall

The SystemOut.log reports that there are no admission control limits for the operation makeCall:

```
CommonUtiliti W com.ibm.soa.parlayx21.thirdparty.utils.CommonUtilities
checkAdmissionControl SOAX0125W:
SOAContextImpl_9.42.170.173_1149781432002_965009116:
ThirdPartyCall_IMS: No admission control limits configured for service
ThirdPartyCall_IMS operation makeCall
```

The likely cause of this problem is typographical error in the configuration of the makeCall operation. To resolve the problem follow these steps:

1. Log into the Integrated Solutions Console.
2. Select **TWSS Administration Console** → **Web Service Platform** → **Single Servers** → **Admission Control.ear** → **Key: soa.SOAConsoleSettings.attribute.name.AdmissionControlMBean.**
3. Select **IMS Third Party Call.**
4. Select **Key: soa.ServiceAttributesMBean.attribute.name.Operations.**
5. Click **New.**
6. In the name and value fields, enter: `makeCall`
7. Click **Add.**
8. Click the newly created **makeCall** link.
9. Set the OperationClusterRateLimit to **5000.**
10. Set the OperationLocalRateLimit to **1000.**
11. Click **OK.**
12. Click **Review.**
13. Click **Save.**

Network Resource Retrieval

The SystemOut.log reports problems with Network Resource Retrieval.

```
PivotHandlerW W com.ibm.ws.webservices.engine.PivotHandlerWrapper
invoke WSWS3734W: Warning: Exception caught from invocation to
com.ibm.ws.webservices.engine.dispatchers.java.JavaBeanDispatcher:
WebServicesFault

faultCode:
{http://www.ibm.com/schema/soa/netres/v1_0/local}NetworkResourceRetriev
alFault
```

```
faultString: com.ibm.soa.sp.netres.NetworkResourceRetrievalFault

faultActor: null

faultDetail:
```

The likely cause of this problem is typographical error in the configuration of the Third Party Call Control service. To resolve the problem follow these steps:

1. Log into the Integrated Solutions Console and select **TWSS Administration Console** → **Web Services** → **Single Servers** → **IMS Third Party Call.ear** → **Third Party Call Web Service.**
2. In the SIP proxy resource specification field type: `SIPProxyResourceSpec`
3. Click **OK.**
4. Click **Review.**
5. Click **Save.**

B.3 IBM WebSphere Group List Server component

This section describes the steps required to install IBM WebSphere Group List Server component. The process describes the minimum installation required for the sample application to successfully execute. The outline of the installation processes consists of the following:

- B.3.1, “Install base binaries” on page 587
- B.3.2, “Create WebSphere Application Server profile” on page 587
- B.3.3, “Configure DB2” on page 589
- B.3.4, “Configure the LDAP directory” on page 591
- B.3.5, “Configure users and groups” on page 592
- B.3.6, “Configure JDBC and data sources” on page 594
- B.3.7, “Tune the Application Server” on page 599
- B.3.8, “Deploy GLS application” on page 603
- B.3.9, “Install the Self Care portlet” on page 606
- B.3.10, “Install the command line interface” on page 609
- B.3.11, “Administration” on page 609

Note: The installation instructions assume that Linux Red Hat Enterprise Linux AS 4.0 Update 3 is being used, WebSphere Application Server 6.1, IBM DB2 Universal Database 8.2 fix pack 4 and IBM Tivoli Directory Server V6.0 are already installed and configured on the machine.

B.3.1 Install base binaries

1. Create **/opt/IBM/GroupListServer** directory.
2. Copy and/or extract all the binaries into **/opt/IBM/GroupListServer** directory.

B.3.2 Create WebSphere Application Server profile

An application server is dedicated to running the GLS. This removes the potential for conflicts between the components for resources and it also allows for performance tuning of the Java Virtual Machine on a product by product basis.

1. Switch to <was_root>/firststeps directory.
2. Start the first steps GUI `./firststeps.sh`.
3. After the wizard opens, perform the following:
   a. Select the profile management tool.
   b. Click **Next.**
   c. Select **Application Server** as the type of WebSphere Server environment to create.
   d. Click **Next.**
   e. Choose **Typical Profile Creation** as profile creation process.
   f. Enable Administrative Security.
      i. Select **Enable Administrative Security.**
      ii. Enter `glsuser` as username for the administrator.
      iii. Enter the **password** for the administrator.
      iv. Confirm the password.
      v. Click **Next.**
   g. Review the profile creation summary.
   h. Click **Create.**

The profile creation complete window will be displayed similar to Figure B-31.

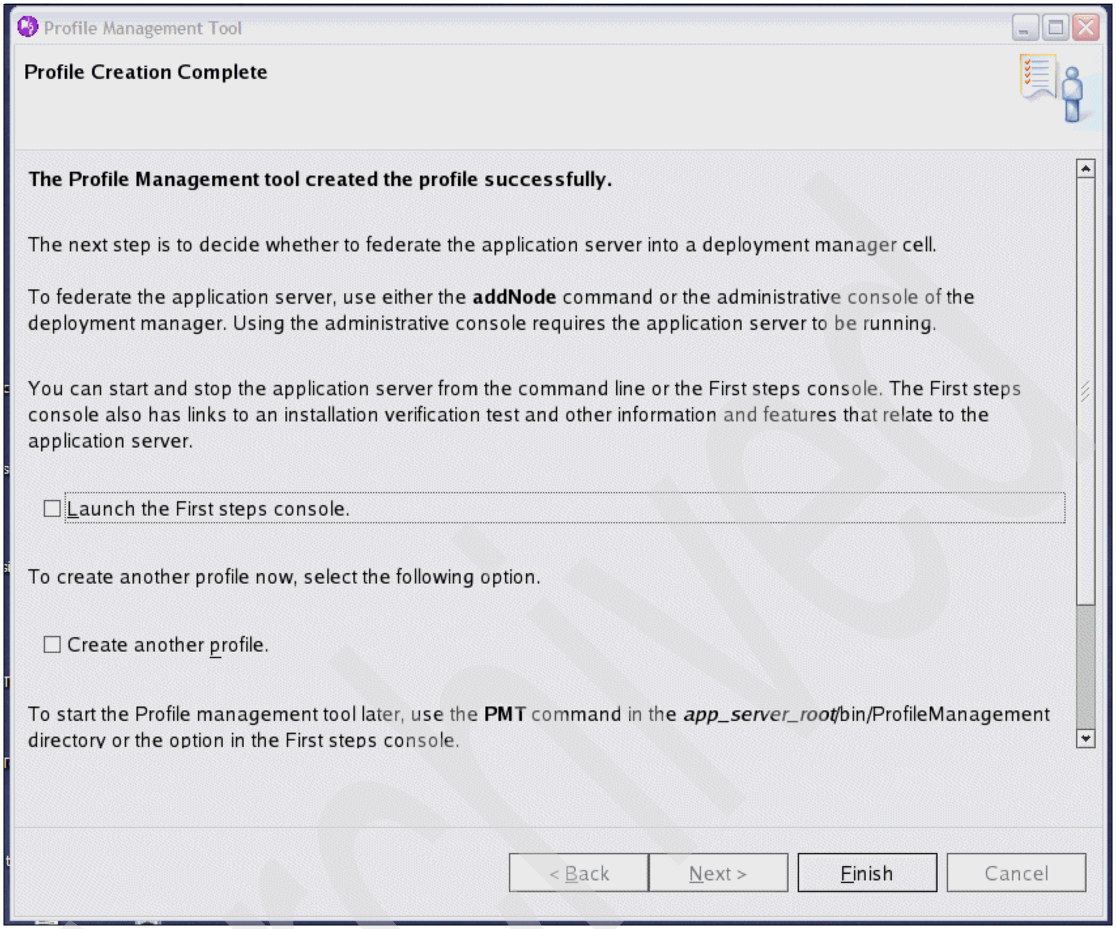


Figure B-31 Profile creation in WebSphere Application Server

4. Verify that the security for the profile is properly configured.

   Login to the administration console. using *glsuser* as the administrator username.

5. In the Integrated Solutions Console navigation panel:

   a. Click **Security** → **Secure administration ,applications, and infrastructure.**

   b. Under Administrative security, select **Enable administrative security.**

   c. Under Application security, select **Enable application security.**

   d. Under Java 2 security, deselect **Use Java 2 security to restrict application access to local resources.**

6. Click **Apply.**
7. Click **Save** changes to the master configuration.

B.3.3 Configure DB2

GLS utilizes database tables to store usage records. The process described in this section is for configuring and creating DB2 databases.

Note: It is assumed that DB2 8.2 with fix pack 4 has been installed.

DB2 databases and tables are created using the script crtsrvDb2.sh. The script requires several parameters that modify the behavior.

Table B-3 GLM610D script arguments

Argument	Description
Database server hostname	This value must be the hostname and domain (for example machine1.ibm.com) **Note:** Localhost should not be used even if the database is hosted locally.
Database server connection port	This value is normally `50000`
Database name	This can be any valued DB2 database name, however `GLM610D` was used for this example
Database alias	This can be any valued DB2 database alias, however `GLM610D` was used for this example
Database locale	For this sample test environment, this value was set to `US`
Database server instance id	In this example, this value was set to db2inst1
Database server instance password	
Database user ID	In this example, the user ID was set to db2inst1
Database user ID password	
Path and file name of DDL file	This value will change depending on the request that is being made
New database directory	This is the file system directory to host the database. In the example this will be `/home/db2inst1/glsdb`

Argument	Description
Database (re)create	This determines if the script will drop the database, and recreate at the beginning of the script.

1. Login to the DB2 server as an administrator.
2. Enter the following command: `su - db2inst1`

   Where, `db2inst1` is the DB2 username.
3. Create a new directory to host the database.

   `mkdir /home/db2inst1/glsdb`
4. Change directory to the location of ctrsrvDb2.sh.

   `cd /opt/IBM/WebSphere/GroupListServer/Database_Setup/DB2`
5. Execute the following command.

   `./crtsrvDb2.sh`
6. Enter the path and filename of the UsageDbDb2.ddl when prompted.
7. A numbered list of parameters will appear.
   a. Enter the number of any value you want to change.
   b. Press Enter.
   c. Enter the new value according to recommendations in Table B-3 on page 589.
   d. Press Enter.

   Figure B-32 on page 591 shows the values that were used for the sample test environment described in this redbook.

```
Press Enter to accept these values and continue, otherwise
  enter the number of the parameter you wish to override.

   1. Database server hostname (dbServer)......:  kpmgyw3.itso.ral.ibm.com
      Recommended Value: fully qualified host name.
   2. Database server connection port (dbPort).:  50000
   3. Database name (dbName)...................:  GLM610D
      Recommended Value: GLM610D
   4. Database alias (dbAlias).................:  GLM610
      Recommended Value: GLM610
   5. Database locale (dbLocale)...............:  US
      Recommended Value: US
   6. Database server instance id (dbInstance).:  db2inst1
   7. Database server instance password (dbInstancPW).:  itso4you
   8. Database userid (dbUser).................:  db2inst1
   9. Database userid password (dbUserPW)......:  db2inst1
  10. Path and file name of DDL file (ddlFile).:  /opt/IBM/WebSphere/GroupListServer/Database/DB2/UsageDbDb2.ddl
  11. New database directory (dbDir)...........:  /home/db2inst1/glsdb
      Recommended Value: /usr/srvspace
      Note: This directory must be created and empty prior to execution.
  12. Database (re)create (dbCreate)...........:  TRUE
      Recommended Value: TRUE
  13. Local Database (dbLocal) ................:  FALSE
      Recommended Value: FALSE
```

Figure B-32 IBM GLS Database creation

8. Verify that USAGEPROPERTIES and USAGERECORDS tables were properly created by entering the following commands:

```
db2 connect to GML610 user db2inst1
db2 list tables
```

B.3.4 Configure the LDAP directory

You must then configure the IBM Tivoli Directory Server by creating a new suffix in the directory. The suffix corresponds to the provider domain under which all groups are created within WebSphere Group List Server.

1. Open the glminit.ldif file in a text editor.
2. Modify the ibm-slapdSuffix for your environment. For example, the value for this test environment is:

```
ibm-slapdSuffix: dc=ibm,dc=com
```

```
db2inst1@kpmgyw3:/opt/ibm/ldap/V6.0/bin
DN: cn=Directory, cn=RDBM Backends, cn=IBM Directory, cn=Schemas, cn=Configuration
changetype: modify
add: ibm-slapdSuffix
ibm-slapdSuffix: dc=ibm,dc=com
~
~
~
                                                          1,1           All
```

Figure B-33 Suffix creation in ITDS

3. Save and close the file.
4. Start IBM Tivoli LDAP directory (ibmslapd in the sbin directory).
5. Run the following command located in the `/bin` directory to create the suffix in the LDAP directory:

   ```
   ldapmodify -D cn=root -w password -i glminit.ldif
   ```

   Where, `cn=root` is the LDAP admin_DN and `password` is the associated password.

B.3.5 Configure users and groups

WebSphere Group List Server require that some users and user groups be created in WebSphere Application Server prior to the installation of the WebSphere Group List Server.

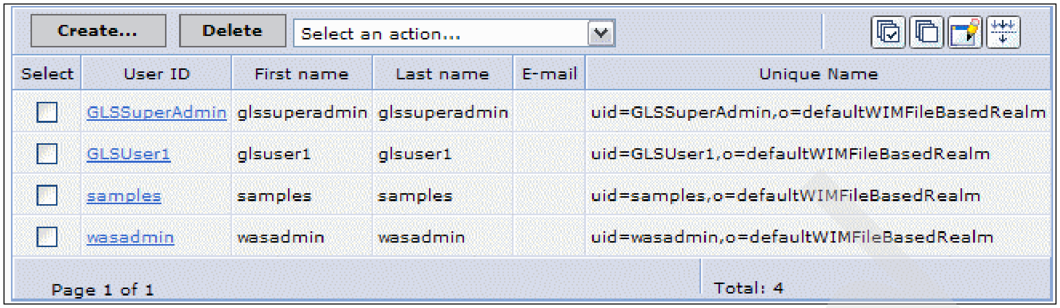
1. Log in to the WebSphere Integrated Solutions Console.

   Enter the user ID and password from B.3.2, "Create WebSphere Application Server profile" on page 587 .
2. In the Integrated Solutions Console navigation panel, click **Users and Groups → Manage users.**
3. Create a user with administrative authority over all groups in the GLS. We recommend using `GLSSuperAdmin` as the User ID.
   a. Click **Create.**
   b. Enter the User ID. For example, type: `GLSSuperAdmin`
   c. Enter the First name and Last name for the user.
   d. Enter the Password for the user.
   e. Confirm the password.
   f. Click **Create.**
   g. A message indicating successful creation of the user will appear.
   h. Click **Close.**
   i. In the Integrated Solutions Console navigation panel:
      i. Click **Users and Groups > Administrative User Roles.**
      ii. Click **Add.**
      iii. Enter the User ID for the user created above.
      iv. Press **Ctrl** and click **Administrator and Configurator for the Role of the user.**
      v. Click **OK.**

4. Create **general users** to use IBM WebSphere Group List Server Component.
   a. Click **Create.**
   b. Enter `GLSUser1` for the User ID.
   c. Enter the user First name and Last name.
   d. Enter the user password.
   e. Confirm the password.
   f. Click **Create.**
   g. A message indicating successful creation of the user will appear.
   h. Click **Close.**
5. Create a **user group** for users accessing IBM WebSphere Group List Server Component.
   a. In the Integrated Solutions Console navigation panel, click **Users and Groups > Manage Groups**.
   b. Click **Create.**
   c. Enter `GLSUsers` for the Group name.
   d. Click **Create.**
   e. A message indicating successful creation of the group will appear.
   f. Click **Close.**
6. Add users to the user group:
   a. Click **group_name -** `GLSUsers.`
   b. Click **Members.**
   c. Click **Add Users.**
   d. Click **Search** to display all available users.
   e. Click the user_id for each user you want to add to the user group.

      The user `GLSUser1` should be in the list.
   f. Click **Add.**
   g. A message indicating successful addition of users of the group will appear.
   h. Click **Close.**

The list of configured users should appear as in Figure B-34 on page 594.



Select	User ID	First name	Last name	E-mail	Unique Name
☐	GLSSuperAdmin	glssuperadmin	glssuperadmin		uid=GLSSuperAdmin,o=defaultWIMFileBasedRealm
☐	GLSUser1	glsuser1	glsuser1		uid=GLSUser1,o=defaultWIMFileBasedRealm
☐	samples	samples	samples		uid=samples,o=defaultWIMFileBasedRealm
☐	wasadmin	wasadmin	wasadmin		uid=wasadmin,o=defaultWIMFileBasedRealm

Figure B-34 User and User Group creation in WebSphere Application Server for GLS

B.3.6 Configure JDBC and data sources

We need to create the data sources in the application server for GLS to be able to access the database. To connect to the database, we must create a Java Authentication and Authorization Service (JAAS) authentication alias for the database, create the JDBC provider, and define the data source using the Integrated Solutions Console.

The topics in this section are:

- "Create authentication alias for the GLS database" on page 594
- "Create JDBC provider" on page 595
- "Map data source to connection to the GLM610 database" on page 596

Create authentication alias for the GLS database

1. Log in to the WebSphere Integrated Solutions Console.

   Enter the user ID and password from B.3.2, "Create WebSphere Application Server profile" on page 587.

2. In the Integrated Solutions Console navigation panel:

   a. Click **Security** → **Secure Administration, applications and infrastructure.**

   a. Expand **Java Authentication and Authorization Service.**

   b. Click **J2C authentication data.**

   c. Click **New.**

3. Enter `GLM610D` in the alias field.

4. Enter `db2inst1` in the user ID field.

   This is the user_id that is used to access the GLS database.

5. Enter the password that corresponds to the user_id in the password field.

6. Click **OK.**
7. Click **Save.**

Create JDBC provider

1. Log in to the WebSphere Integrated Solutions Console.

   Enter the user ID and password from B.3.2, "Create WebSphere Application Server profile" on page 587.
2. In the Integrated Solutions Console navigation panel, click **Resources** → **JDBC** → **JDBC providers.**
3. Expand Scope.
4. Select node_name from the drop-down list.
5. Click **New.**
6. Select **DB2** as the database type from the drop-down list.
7. Select **DB2 Universal JDBC Driver Provider** as the Provider type for your database.
8. Select **Connection pool data source** as the Implementation type.
9. Click **Next.**
10. Enter the values for the class paths for your database.
    a. */home/db2inst1/sqllib/java* for jar file location
    b. */home/db2inst1/sqllib/lib* for native library
11. Click **Next.**
12. Verify all values are correct.
13. Click **Finish.**
14. Click **Save** to save the changes to the master configuration.

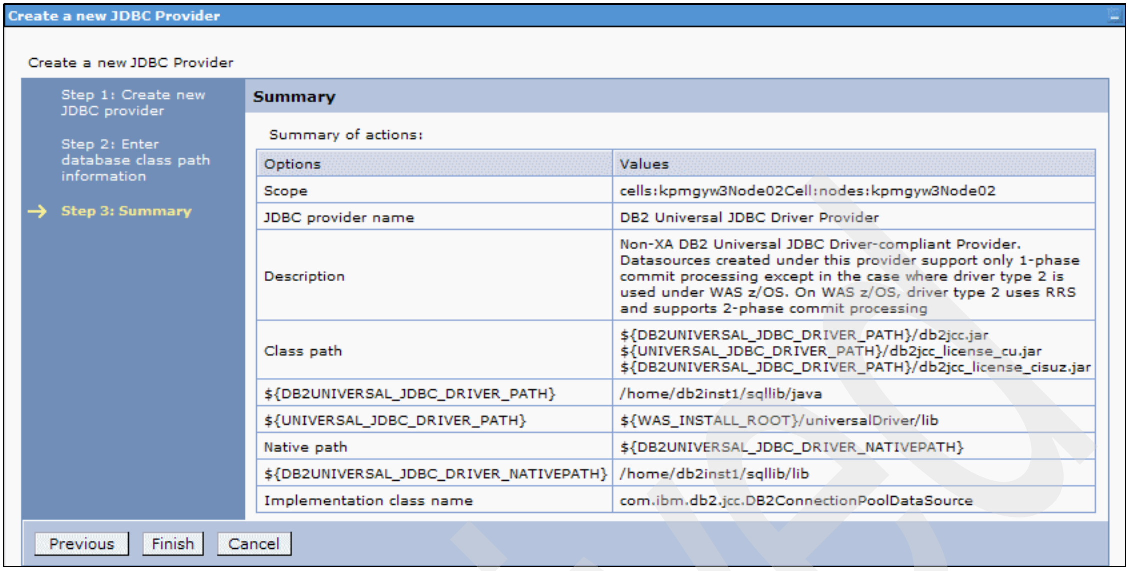


Figure B-35 Create JDBC provider

Map data source to connection to the GLM610 database

1. Log in to the WebSphere Integrated Solutions Console.

   Enter the user ID and password from B.3.2, “Create WebSphere Application Server profile” on page 587.

2. In the Integrated Solutions Console navigation panel:

   a. Click **Resources → JDBC → JDBC Providers.**

   b. Click `jdbc_provider` to open the properties for the JDBC provider you want to configure the data source for.

   c. Click **Data sources.**

   d. Click **New.**

3. Enter `GLMUsageRecords` in the Data source name field.
4. Enter `jdbc/GLMUsageRecordsDS` in the JNDI name field.
5. Select `<nodename>/GLM610D` from the Component-managed authentication alias drop-down list.
6. Click Select **DB2 Universal JDBC Driver Provider** as JDBC provider.
7. Enter `GLM610` as the database_name in the Database name field.
8. Enter `4` in the Driver type field to specify the connectivity type of the data source.

This value corresponds with the driver type property in the data source class.

9. Enter `localhost` as server_name in the Server name field.
10. Enter `50000` in the Port number field.

    This value corresponds with the port number property in the data source class.
11. Select **Use this data source in container managed persistence (CMP)** check box.
12. Click **Next.**
13. Verify the values are correct.
14. Click **Finish.**
15. Click **Save** to save the changes to the master configuration.

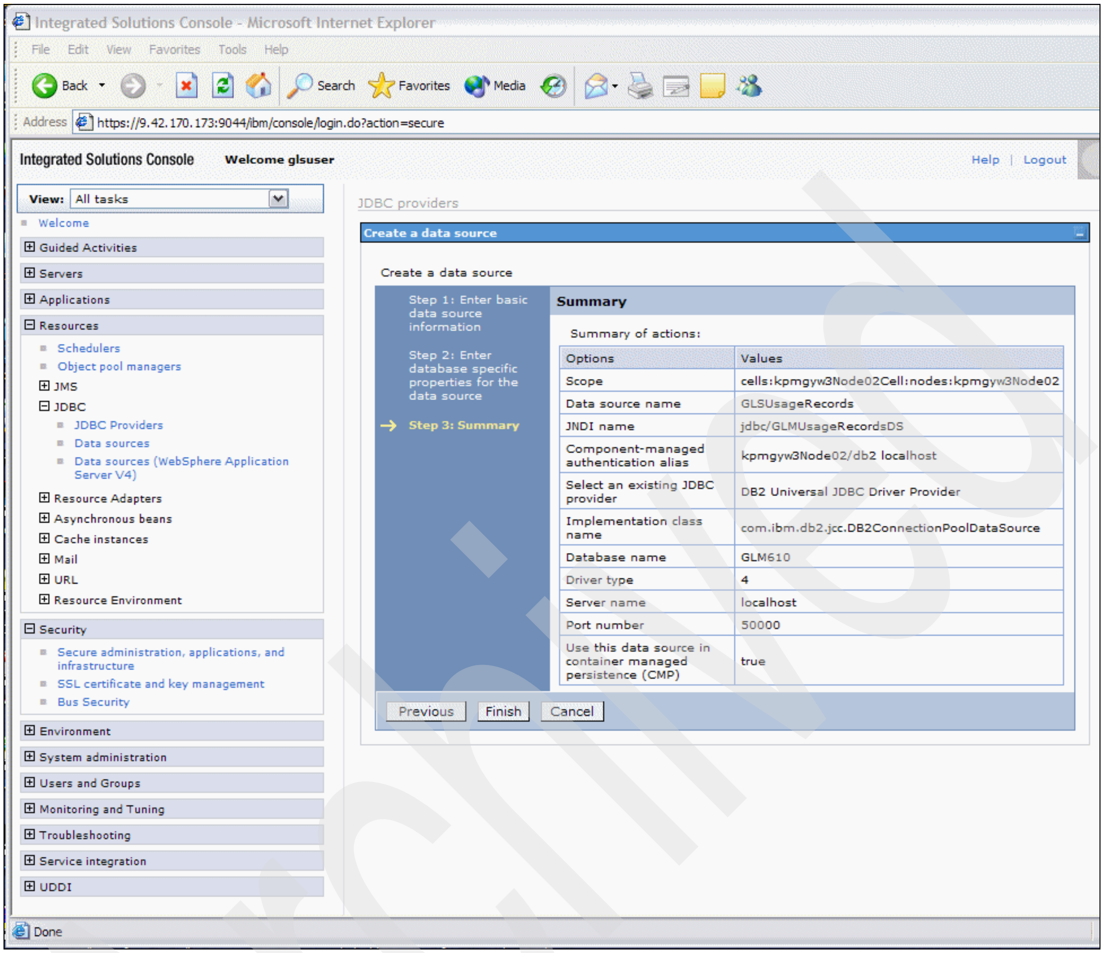


Figure B-36 Creation of a data source in WebSphere Application Server

16. Test the connection to DB2 database.
    a. Select the associated check box for the data source.
    b. Click Test connection.
    c. A screen similar to Figure B-37 will be displayed.

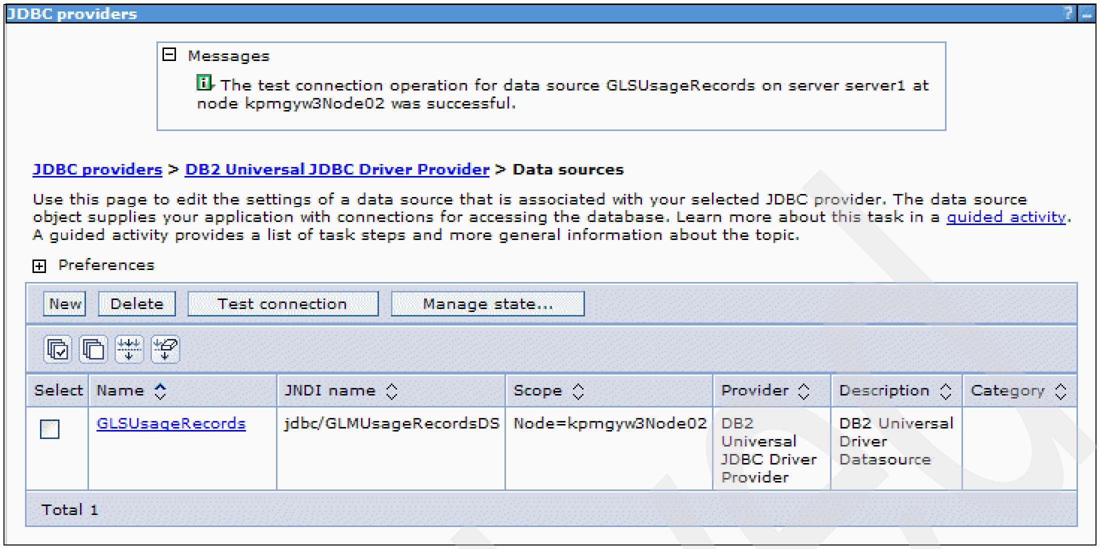


Figure B-37 Test the connection with DB2

B.3.7 Tune the Application Server

We need to configure the GLS resource environment provider.

The topics in this section are:

- "Configure glmconfig. properties file" on page 599
- "Restart the application server" on page 601
- "Configure cache instances" on page 601

Configure glmconfig. properties file

To do so we start by modifying the `glmconfig.properties` file.

1. Open `glmconfig. properties` located in the directory: `<install_path>/IBM_Group_List_Manager/Server_Application/Configuration`
2. Modify the following properties.
   – repository URL (if necessary)
   – repositoryAdmin

     For example, `cn=root` and `repositoryAdminPwd` for the LDAP admin password
   – superAdmin

     The username for SuperAdmin user

– superAdminPswd

  The password for SuperAdmin user

– providerDomain

  The domain name, for example `ibm.com`

– xcapRootURL

  This URL is used to retrieve XCAP document. The value depends on the application server instance.

3. Save and close the file.
4. Open a command prompt .
5. Switch to the `was_profile_root/bin` directory.
6. Run the command:

```
./wsadmin.sh -username user_name -password password
-wsadmin_classpath script_path/GLMUtil.jar -f
script_path/CreateGLMProps.jacl script_path Node:node_name locale
```

   Where:

– user_name

  Represents your WebSphere Application Server administrator user ID (`glsuser` for this sample test environment)

– password

  Represents the password associated with user_name

– script_path

  Represents the path for the GLS configuration files

– node_name

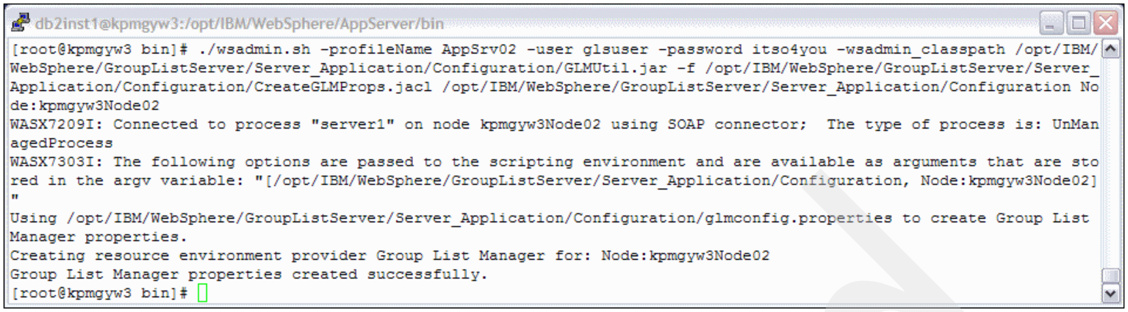
  Represents the name of the node where WebSphere Group List Server is deployed

– locale

  Represents the *optional* parameter that you should specify if you are going to use a translated version of glmconfig.properties to create the WebSphere Group List Server resource environment provider.

Important: Do not use relative paths.

Figure B-38 on page 601 shows a sample screen from running the command.



```
db2inst1@kpmgyw3:/opt/IBM/WebSphere/AppServer/bin
[root@kpmgyw3 bin]# ./wsadmin.sh -profileName AppSrv02 -user glsuser -password itso4you -wsadmin_classpath /opt/IBM/
WebSphere/GroupListServer/Server_Application/Configuration/GLMUtil.jar -f /opt/IBM/WebSphere/GroupListServer/Server_
Application/Configuration/CreateGLMProps.jacl /opt/IBM/WebSphere/GroupListServer/Server_Application/Configuration No
de:kpmgyw3Node02
WASX7209I: Connected to process "server1" on node kpmgyw3Node02 using SOAP connector;  The type of process is: UnMan
agedProcess
WASX7303I: The following options are passed to the scripting environment and are available as arguments that are sto
red in the argv variable: "[/opt/IBM/WebSphere/GroupListServer/Server_Application/Configuration, Node:kpmgyw3Node02]
"
Using /opt/IBM/WebSphere/GroupListServer/Server_Application/Configuration/glmconfig.properties to create Group List
Manager properties.
Creating resource environment provider Group List Manager for: Node:kpmgyw3Node02
Group List Manager properties created successfully.
[root@kpmgyw3 bin]#
```

Figure B-38 Configuration of the GLS resource environment provider

Restart the application server

1. To stop the server, run the following command from the application server profile:

   `was_profile_root/bin/stopServer.sh server_name -username user_name -password password`

   Where:

   – server_name

     Is name of the application `server`

   – user_name

     Is the WebSphere Application Server administrator user ID. This parameter is required if security has been enabled, which is the case for the GLS application server.

   – `password`

     Is the password associated with user_name above. This parameter is required if security is enabled.

2. Restart the server by running the following command:

   `was_profile_root/bin/startServer.sh server_name`

   Where:

   – server_name

     Is name of the application `server`

Configure cache instances

The WebSphere Group List Server uses two cache instances. One is for group objects containing member and attribute information, and the other is for group access control lists for authorization. Configure cache instances using the following steps:

1. Log in to the WebSphere Integrated Solutions Console.
2. Enter the user ID and password from B.3.2, “Create WebSphere Application Server profile” on page 587.
3. In the Integrated Solutions Console navigation panel:
   a. Click **Resources** → **Cache instances** → **Object cache instances.**
   b. Select **cell_name** for the scope.
4. Create the `GLMGroupCache`.
   a. Click **New.**
   b. Select `cell_name` for the scope.
   c. Enter `GLMGroupCache` for the Name.
   d. Enter `services/cache/com/ibm/glm/groupcache` for the JNDI name.
   e. Enter `10000` for the Cache size.
   f. **Deselect** Dependency ID Support.
   g. Click **OK.**
5. Create the `GLMGroupACLsCache`.
   a. Click **New.**
   b. Select `cell_name` for the scope.
   c. Enter `GLMGroupACLsCache` for the Name.
   d. Enter `services/cache/com/ibm/glm/groupaclscache` for the JNDI name.
   e. Enter `10000` for the Cache size.
   f. **Deselect** Dependency ID Support.
   g. Click **OK.**
6. Create the `GLMXcapXmlCache`.
   a. Click **New.**
   b. Select `cell_name` for the scope.
   c. Enter `GLMXcapXmlCache` for the Name.
   d. Enter `services/cache/com/ibm/glm/xcapxmlcache` for the JNDI name.
   e. Enter `10000` for the Cache size.
   f. **Deselect** Dependency ID Support.
   g. Click **OK.**
7. Create the `GLMAdminDomainsCache`.
   a. Click New.

b. Select `cell_name` for the scope.

c. Enter `GLMAdminDomainsCache` for the Name.

d. Enter `services/cache/com/ibm/glm/admindomainscache` for the JNDI name.

e. Enter `100` for the Cache size.

f. **Deselect** Dependency ID Support.

g. Click **OK**.

8. Click **Save** to save changes to the master configuration.

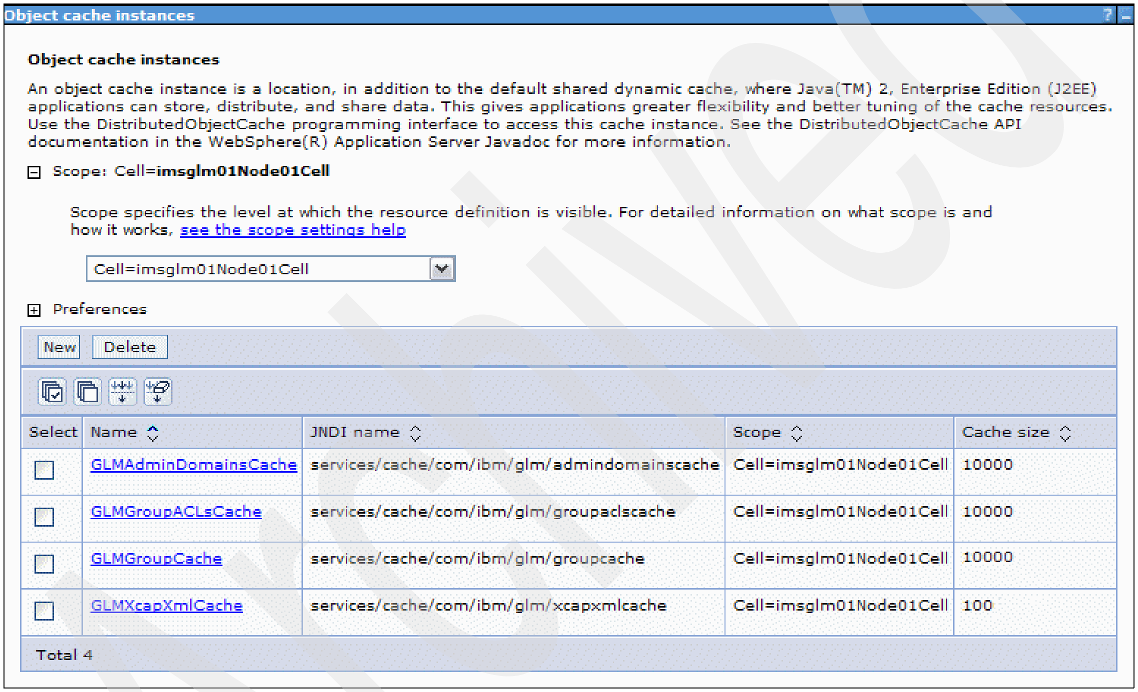


Figure B-39 Configuring cache instances

B.3.8 Deploy GLS application

The IBM GLS application is installed as an enterprise application in WebSphere Application Server.

The topics in this section are:

- "Install GroupListMgr.ear" on page 604
- "Assign access to users" on page 604
- "Define user RunAs role" on page 605

- “Start the GLS Application” on page 606

Install GroupListMgr.ear

1. Log in to the WebSphere Integrated Solutions Console.
2. Enter the user ID and password used during the creation of the GLS server instance in Appendix A, “Installing the application development environment” on page 499.
3. In the Integrated Solutions Console navigation panel:
   a. Click **Applications** → **Install new Application.**
   b. Click **Browse** to locate the `GroupListMgr.ear` file on your system.
4. Click **Next** → **Next** → **Next** → **Finish.**
5. Click **Save** to the master configuration.

Assign access to users

1. Click **Applications** → **Enterprise Applications.**
2. Click **Group List Manager.**
3. Under **Detail Properties,** click **Security role to user/group mapping.**
4. Assign `GLMConfigurator` role.
   a. Select the check box associated with `GLMConfigurator` role.
   b. Click **Look up users.**
   c. Verify `GLMConfigurator` appears in the list of roles near the beginning of the page.

   > **Note:** Entering asterisk(*), will return all users in the search results, unless the number of users exceeds the value in the limit field.

   d. Click **Search.**
   e. Click the name of the user who you assigned configurator authority to (this should be `GLSConfigUser`).
   f. Click **>>** to move the user to the Selected column.
   g. Click **OK.**
   h. The selected user should be listed in the Mapped users column on the GLMConfigurator row.
5. Assign `GLMAdapterClient` role.
   a. Select the check box associated with `GLMAdapterClient` role.

b. Click **Look up users.**

c. Verify `GLMAdapterClient` appears in the list of roles near the top of the page.

> **Note:** Entering asterisk(*), will return all users in the search results, unless the number of users exceeds the value in the limit field

d. Click **Search.**

e. Click the name of the user who you assigned operator authority to (this should be `GLSAdapterUser`).

f. Click >> to move the user or group to the Selected column.

g. Click **OK.**

h. The user that you selected is listed in the Mapped users column on the GLMAdapterClient row.

6. Assign `GLMXcapUser` role.

   a. Select the check box associated with `GLMXcapUser` role.

   b. Click **Look up groups.**

   c. Verify `GLMXcapUser` appears in the list of roles near the top of the page.

   > **Note:** Entering asterisk(*), will return all groups in the search results unless the number of groups exceeds the value in the limit field.

   d. Click **Search.**

   e. Click the name of the WebSphere Group List Server group (this should be `GLSUsers`).

   f. Click **>>** to move the group to the Selected column.

   g. Click **OK.**

   h. The group you selected is listed in the Mapped groups column on the GLMXcapUser row.

7. Click **OK** to save all of the security role to user and group mappings and return to **Enterprise Applications** → **GroupListMgr.**

Define user RunAs role

The GroupListMgr EAR contains predefined RunAs roles. In this section, the user is identified using the configurator authority.

1. In the Integrated Solutions Console navigation panel:

a. Click **Applications** → **Enterprise Applications.**

b. Click **GroupListMgr.**

c. Click **User RunAs** roles.

2. Under **Detail Properties**:

   a. Enter the username and password for the user with configurator authority, such as `GLSSuperAdmin`.

   b. Select the check box that corresponds to the `GLMConfigurator` role.

   c. Click **Apply.**

   d. The username that you entered is listed in the User name column on the GLMConfigurator row.

   e. Click **OK** to save the role assignment and return to **Enterprise Applications** → **GroupListMgr.**

   f. Click **Save** to save to the master configuration.

Start the GLS Application

1. In the Integrated Solutions Console navigation panel, click **Applications** → **Enterprise Applications.**
2. Select **Group List Server.**
3. Click **Start.**
4. The GLS application should be started successfully.
5. Check the SystemOut log file for any errors.

B.3.9 Install the Self Care portlet

The Group List Server Self Care portlet provides group and member administration through a graphical user interface. The portlet is installed in WebSphere Application Server as an enterprise application.

The topics in this section are:

- "Install GLMAdmin.ear" on page 606
- "Assign access to users" on page 607
- "Start the application" on page 607
- "Verify installation" on page 608

Install GLMAdmin.ear

1. Log in to the WebSphere Integrated Solutions Console.
2. Enter the user ID and password used for the creation of the application Server for GLS.

3. In the WebSphere Integrated Solutions Console navigation panel, click **Applications** → **Install new Application.**
4. Click **Browse** to locate the `GLMAdmin.war` file on your system.
5. Enter a context root for the portlet.

   The context root will be combined with the defined servlet mapping, `GLMAdmin`, to compose the full URL that users will enter to access the portlet. For example, if the context root is `MyContext`, the URL is `http://host:port/MyContext/GLMAdmin`.
6. Click **Next** → **Next** → **Next** → **Finish.**
7. Click **Save** to save to the master configuration.

Assign access to users

1. In the WebSphere Integrated Solutions Console navigation panel, click **Applications** → **Enterprise Applications.**
2. Click **GLMAdmin.**
3. Under **Detail Properties.**
   a. Click **Security role to user/group mapping.**
   b. Select **GLMAdminPortlet** role.
   c. Click **Look up groups.**
   d. Search for the group whose members are the WebSphere Group List Server users who should have access to Group List Management Self Care. This is the group that you created earlier in the installation.
   e. Click the user group you want to provide access to.
   f. Click **>>** to add the group to the Selected column.
   g. Click **OK.**
4. The members of the group that you selected is listed in the Mapped users column on the GLMAdminPortlet row.

Start the application

1. In the WebSphere Integrated Solutions Console navigation panel:
   a. Click **Applications** → **Enterprise Applications.**
   b. Select the check box associated with the `GLMAdmin`.
   c. Click **Start.**
   d. The Application Status column should indicate a Started status.

Verify installation

Verify **Group List Management Self Care** is correctly installed by performing the following steps.

1. Upon opening a browser, enter the URL: `http://hostname:port/context_root/GLMAdmin/`

   Where:

   – hostname

     Is the server name that runs the GLS application

   – context_root

     Is the context root specified in Step 5., “Enter a context root for the portlet.” on page 607 previously.

2. When you access this URL, you will be prompted to enter a username and password.

   a. Enter the username and password of a user who is the member of the group mapped to the `GLMAdminPortlet` role (this should be `GLSUser1`).

   b. Then you should see the maim GLS administration portlet menu as shown in Figure B-40.

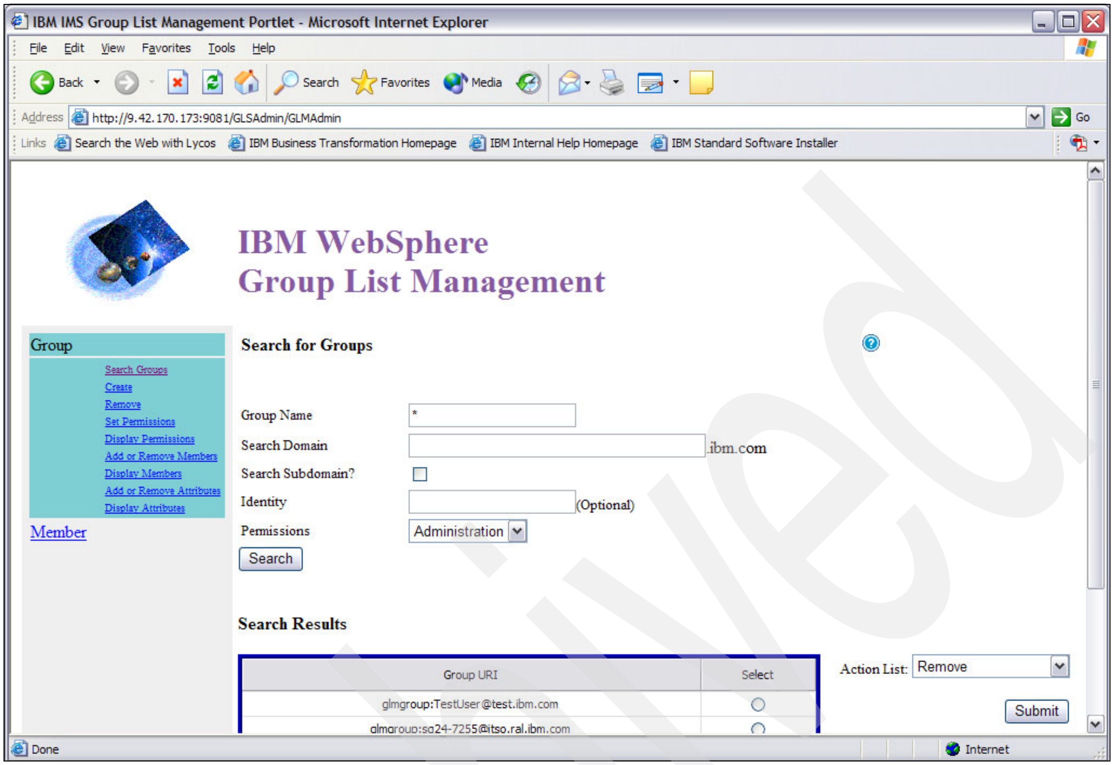


Figure B-40 GLS Self Care portlet

B.3.10 Install the command line interface

The command line interface is used to access WebSphere Group List Server through the XDM interface.

1. Verify that you have `glmcli.zip` file which is needed for the installation.

   You can find it in the directory /opt/IBM/Group_List_Server/Tools.

2. Create a new directory `glmcli` in /opt/IBM/Group_List_Server/Tools.
3. Extract `glmcli.zip` to the directory you created.
4. Run the chmod command (chmod 755 *.*) on the files so you will have sufficient authority to run the scripts.

B.3.11 Administration

You can administer GLS with either of two interfaces, the Group List Management Self Care portlet or the command line interface.

Group List Management Self Care portlet

Group List Management Self Care portlet provides group and member administration through a graphical user interface. Additionally, you can use the portlet to add and delete groups, add and delete members, delegate and remove permissions, query both group and member memberships, and to add and remove user-defined attributes.

Command line interface

The command line interface is used to bulk load groups. You can also use command line interface to add, delete, and query group memberships. Example B-1 shows example of how to add new groups in GLS using the command line interface.

Example: B-1 Command line instruction for adding groups to GLS

```
./xcap_put.sh -user GLSUSer1 -password itso4you -filename ./redbook.xml
-content_type application/resource-lists+xml
http://localhost:9081/services/resource-lists/users/GLSUser1/RedbookGro
up.xml

***********************************************************************
        GLM XCAP Client Interface       - PUT COMMAND
***********************************************************************
Response Status was: 201
Response Status Message: Created
Content-Type: application/xcap-diff+xml; charset=UTF-8
```

Example B-2 on page 610 shows how to retrieve existing groups from GLS using the command line interface.

Example: B-2 Command line instruction for retrieving existing groups from GLS

```
./xcap_get.sh -user GLSUser1 -password itso4you -filename ./redbook.xml
http://localhost:9081/services/resource-lists/users/GLSUser1/RedbookGro
up.xml

**********************************************************************
        GLM XCAP Client Interface       - GET COMMAND
**********************************************************************
Response Status was: 200
Response Status Message: OK
Content-Type: application/resource-lists+xml; charset=UTF-8
```

Example B-3 shows how to delete existing groups from GLS with the command line interface.

Example: B-3 Command line instruction for deleting existing groups from GLS

```
./xcap_delete.sh -user GLSUser1 -password itso4you
http://localhost:9081/services/resource-lists/users/GLSUser1/RedbookGro
up.xml
***********************************************************************
        GLM XCAP Client Interface       - DELETE COMMAND
***********************************************************************

Response Status was: 200
Response Status Message: OK
Content-Type: text/html
```

B.4 IBM WebSphere Presence Server component

This section describes the steps required to install IBM WebSphere Presence Server component (PS). The process describes the minimum installation required for the sample application to successfully execute. The outline of the installation processes consist of the following:

- B.4.1, "Install base binaries" on page 611
- B.4.2, "Create WebSphere Application Server profile" on page 611
- B.4.3, "Configure DB2" on page 612

> **Note:** The installation instructions assume that Linux Red Hat Enterprise Linux AS 4.0 Update 3 is being used, WebSphere Application Server 6.1 and IBM DB2 Universal Database 8.2 fix pack 4 are already installed and configured on the machine.

B.4.1 Install base binaries

1. Create **/opt/IBM/PresenceServer** directory.
2. Copy and/or extract all the binaries into **/opt/IBM/PresenceServer** directory.

B.4.2 Create WebSphere Application Server profile

An application server is dedicated to PS. This removes the potential for conflicts between the components for resources and it also allows for performance tuning of the Java Virtual Machine on a product by product basis.

1. Switch to <was_root>/firststeps directory.
2. Start the first steps GUI `./firststeps.sh.`
3. After the wizard opens, perform the following:
   a. Select the profile management tool.
   b. Click **Next.**
   c. Select **Application Server** as the type of WebSphere Server environment to create.
   d. Click **Next.**
   e. Choose **Typical Profile Creation** as profile creation process.
   f. Deselect **Enable Administrative Security.**
   g. Click **Next.**
   h. Review the profile creation summary.
   i. Click **Create**.

B.4.3 Configure DB2

PS utilizes DB tables for storing usage records. The following steps describe the process for creating and configuring DB2.

> **Note:** It is assumed that DB2 8.2 with fix pack 4 has been installed.

A new database `SOA610D` needs to be created for PS. DB2 databases and tables are created using a script called WPSSetDB2Tables.sh. The script requires several parameters that modifies the behavior.

Table B-4 SOA610D script arguments

Argument	Description
Database server hostname	This value must be the hostname and domain (for example machine1.ibm.com) **Note:** Localhost should not be used even if the database is hosted locally.
Database server connection port	This value is normally `50000`
Database name	This can be any valued DB2 database name, however `SOA610D` was used for this example
Database alias	This can be any valued DB2 database alias, however `SOA610D` was used for this example

Argument	Description
Database locale	For this sample test environment, this value was set to `US`
Database server instance id	In this example, this value was set to db2inst1
Database server instance password	
Database user ID	In this example, the user ID was set to db2inst1
Database user ID password	
Path and file name of DDL file	This value will change depending on the request that is being made
New database directory	This is the file system directory to host the database. In the example this will be `/home/db2inst1/presencedb`
Database (re)create	This determines if the script will drop the database, and recreate at the beginning of the script.

1. Login to the DB2 server as an administrator.
2. Enter the following command: `su - dbinst1`

   Where, `dbinst1` is the DB2 username
3. Create a new directory to host the database.

   ```
   mkdir /home/db2inst1/presencedb
   ```
4. Change directory to the location of WPSSetDB2Tables.sh.

   ```
   cd /opt/IBM/WebSphere/IBMWebSpherePresenceServer/Database_Setup/DB2
   ```
5. Execute the following command.

   ```
   ./WPSSetDB2Tables.sh
   ```
6. Enter the path and filename of the `CreateDB2Tables.ddl` when prompted.
7. A numbered list of parameters will appear.
   a. Enter the number of any value you want to change.
   b. Press Enter.
   c. Enter the new value according to recommendations in Table B-4 on page 612.
   d. Press Enter.

Figure B-41 shows the values that were used for the sample test environment described in this redbook.

```
db2inst1@kpmgyw3:/opt/IBM/WebSphere/IBMWebSpherePresenceServer/Database_Setup/DB2
11
    Enter the path to the directory to create database in:
    The db user must have permission to create and write in this directory.
/home/db2inst1/presencedb
Usage: crtsrvDb2.sh dbServer dbPort dbName dbAlias dbLocale
                    dbInstance dbInstancePW dbUser dbUserPW
                    ddlFile dbDir dbCreate menuflag
These values will be used to create the database.
Each value must be filled in.
Press Enter to accept these values and continue, otherwise
  enter the number of the parameter you wish to override.
The menuflag should be set to TRUE to display the menu.

   1. Database server hostname (dbServer)......:  kpmgyw3.itso.ral.ibm.com
   2. Database server connection port (dbPort).:  50000
   3. Database name (dbName)...................:  SOA610D
      Recommended Value: SOA610D
   4. Database alias (dbAlias).................:  SOA610
      Recommended Value: SOA610
   5. Database locale (dbLocale)...............:  US
      Recommended Value: US
   6. Database server instance id (dbInstance).:  db2inst1
   7. Database server instance password (dbInstancPW).:  itso4you
   8. Database userid (dbUser).................:  db2inst1
   9. Database userid password (dbUserPW)......:  itso4you
  10. Path and file name of DDL file (ddlFile).:  /opt/IBM/WebSphere/IBMWebSpherePresenceServer/Database_Setup/DDL/CreateDB2Tables.dd
  11. New database directory (dbDir)...........:  /home/db2inst1/presencedb
      Recommended Value: /usr/srvspace
  12. Database (re)create (dbCreate)...........:  TRUE
      Recommended Value: TRUE
```

Figure B-41 IBM PS Database creation

> **Important:** You should enter the same values you entered for the database name and alias you entered when creating the WebSphere Presence Server tables. For the database (re)create option, you must enter FALSE since the database has already been created.

8. Create the usage record tables.
9. Enter the path and filename of the CreateUsageRecordTables.ddl when prompted.
10. A numbered list of parameters will appear.
11. Verify the database was created properly by typing the following command.

```
db2 connect to SOA610 user db2inst1
db2 list tables
```

The list of PS DB2 tables will be displayed as in Figure B-42.

```
[db2inst1@kofrzoy DB2]$ db2 connect to SOA610 user db2inst1
Enter current password for db2inst1:

   Database Connection Information

 Database server        = DB2/LINUX 8.2.4
 SQL authorization ID   = DB2INST1
 Local database alias   = SOA610

[db2inst1@kofrzoy DB2]$ db2 list tables

Table/View                      Schema          Type  Creation time
------------------------------- --------------- ----- --------------------------
CONFIG                          DB2INST1        T     2006-06-08-18.41.29.382858
FULLDOC                         DB2INST1        T     2006-06-08-18.41.29.473028
PUBLISH                         DB2INST1        T     2006-06-08-18.41.29.420280
USAGEPROPERTIES                 DB2INST1        T     2006-06-08-18.45.25.356610
USAGERECORDS                    DB2INST1        T     2006-06-08-18.45.25.087998
XPROVIDERS                      DB2INST1        T     2006-06-08-18.41.29.664193

  6 record(s) selected.
```

Figure B-42 Presence Server DB2 tables

B.5 Create the Service Integration Bus and bus members

WebSphere Presence Server requires a Service Integration Bus. To create the Service Integration Bus do the following:

1. Log in to the WebSphere Integrated Solutions Console.
2. Since the security has not been enabled, there is no need to enter a user ID and password.
3. In the WebSphere Integrated Solutions Console navigation panel:
   a. Click **Service Integration** → **Buses.**
   b. Click **New.**
   c. Enter `PS_bus` for the name of the new bus.
   d. Deselect **Bus security.**
   e. Click **Next.**
   f. Click **Finish.**
4. Click **Save** to save changes to the master configuration.

Add bus members

1. In the WebSphere Integrated Solutions Console navigation panel.
   a. Click **Service integration → Buses.**
   b. Click **PS_bus.**
   c. Under **Topology**, click **Bus members.**
   d. Click **Add.**
   e. Click **Server.**
   f. Select the appropriate server from the drop-down list.
   g. Click **Next.**
   h. Click **File store.**
   i. Click **Next.**
   j. Click **Next.**
   k. Click **Finish.**
2. Click **Save** to save changes to the master configuration.

Define the JMS factories

1. In the WebSphere Integrated Solutions Console navigation panel, click **Resources → JMS → JMS providers.**
2. Click **Default messaging provider** for the correct node you want to configure.
3. Under **Additional Properties**, click **Topic connection factories**.
4. Click **New.**
5. Enter `PresenceTCF` in the Name field.
6. Enter `jms/presenceTCF` in the **JNDI** name field.
7. Select `PS_bus` for the **Bus name.**
8. Click **Apply.**
9. Click **Save** to save changes to the master configuration.

Define the JMS topic

1. In the WebSphere Integrated Solutions Console navigation panel, click **Resources → JMS → JMS providers**.
2. Click **Default messaging provider** for the correct node you want to configure.
3. Under **Additional Properties**, click **Topics.**

4. Click **New.**
5. Enter `PresencePublishT` in the Name field.
6. Enter `jms/presencePublishT` in the **JNDI** name field.
7. Select `PS_bus` for the **Bus name.**
8. Select **Default.Topic.Space** for the Topic space.
9. Click **Apply.**
10. Click **Save** to save changes to the master configuration.

Define the JMS activation specifications

1. In the WebSphere Integrated Solutions Console navigation panel, click **Resources → JMS → JMS providers.**
2. Click **Default messaging provider** for the correct node you want to configure.
3. Under **Additional Properties**, click **Activation specifications**.
4. Click **New.**
5. Enter `TAS` in the Name field.
6. Enter `jms/tas` in the **JNDI** name field.
7. Select `Topic` for the Destination type.
8. Enter `jms/presencePublishT` for the Destination JNDI name.
9. Select `PS_bus` for the Bus name.
10. Select `Auto-acknowledge` for the Acknowledge mode.
11. Select `Nondurable` for the Subscription durability.
12. Click **Apply.**
13. Click **Save** to save changes to the master configuration.

Restart the PS application server

1. To stop the server, run the following command from the application server profile:

   ```
   was_profile_root/bin/stopServer.sh server_name
   ```

   Where:

   `server_name` is the name of the application server (this should be `server1`).
2. To restart the server, run the following command:

   ```
   was_profile_root/bin/startServer.sh server_name
   ```

Where:

`server_name` is the name of the application server.

B.5.1 Configure JDBC and data source

We need to create the data sources in the application server for the Presence Server to be able to access the database. To connect to the database we must perform the following:

- Create a JAAS authentication alias for the database
- Create the JDBC provider
- Define the data source using the Integrated Solutions Console

Create an authentication alias

1. Log in to the WebSphere Integrated Solutions Console.
2. In the navigation panel, click **Security** → **Secure Administration, applications and infrastructure.**
3. Expand **Java Authentication and Authorization Service.**
4. Click **J2C authentication data.**
5. Click **New.**
6. Enter `SOA610D` in the alias field.
7. Enter `db2inst1` in the user ID field; this is the `user_id` used to access the PS database.
8. Enter the password that corresponds to `user_id` in the password field.
9. Click **OK.**
10. Click **Save.**

Create a JDBC provider

1. Log in to the WebSphere Integrated Solutions Console.
2. In the navigation panel, click **Resources** → **JDBC** → **JDBC providers.**
3. Expand Scope.
4. Select node_name from the drop-down list.
5. Click **New.**
6. Select **DB2** as the database type from the drop-down list.
7. Select **DB2 Universal JDBC Driver Provider** as the Provider type for your database.
8. Select **Connection pool data source** for the Implementation type.

9. Click **Next.**
10. Enter `/home/db2inst1/sqllib/java` for the database class path.
11. Click **Next.**
12. Verify all values are correct.
13. Click **Finish.**
14. Click **Save** to save the changes to the master configuration.

Define data source

1. Log in to the WebSphere Integrated Solutions Console.
2. In the navigation panel, click **Resources** → **JDBC** → **JDBC Providers.**
3. Click **jdbc_provider** to open the properties for the JDBC provider to configure.
4. Click **Data sources.**
5. Click **New.**
6. Enter `WPS DB DataSource` in the Data source name field.
7. Enter `jdbc/db` in the JNDI name field.
8. Select `<nodename>/SOA610D` from the Component-managed authentication alias drop-down list.
9. Select **DB2 Universal JDBC Driver Provider** as JDBC provider.
10. Click **Next.**
11. Enter `SOA610` as the database_name in the Database name field.
12. Enter `4` in the Driver type field to specify the connectivity type of the data source.

    This value corresponds with the driver type property in the data source class.
13. Enter `localhost` as server_name in the Server name field.
14. Enter `50000` as the port_number in the Port number field.

    This value corresponds with the port number property in the data source class.
15. Select the **Use this data source in container managed persistence (CMP)** check box.
16. Click **Next.**
17. Verify the values are correct.
18. Click **Finish.**
19. Click **Save** to save the changes to the master configuration.

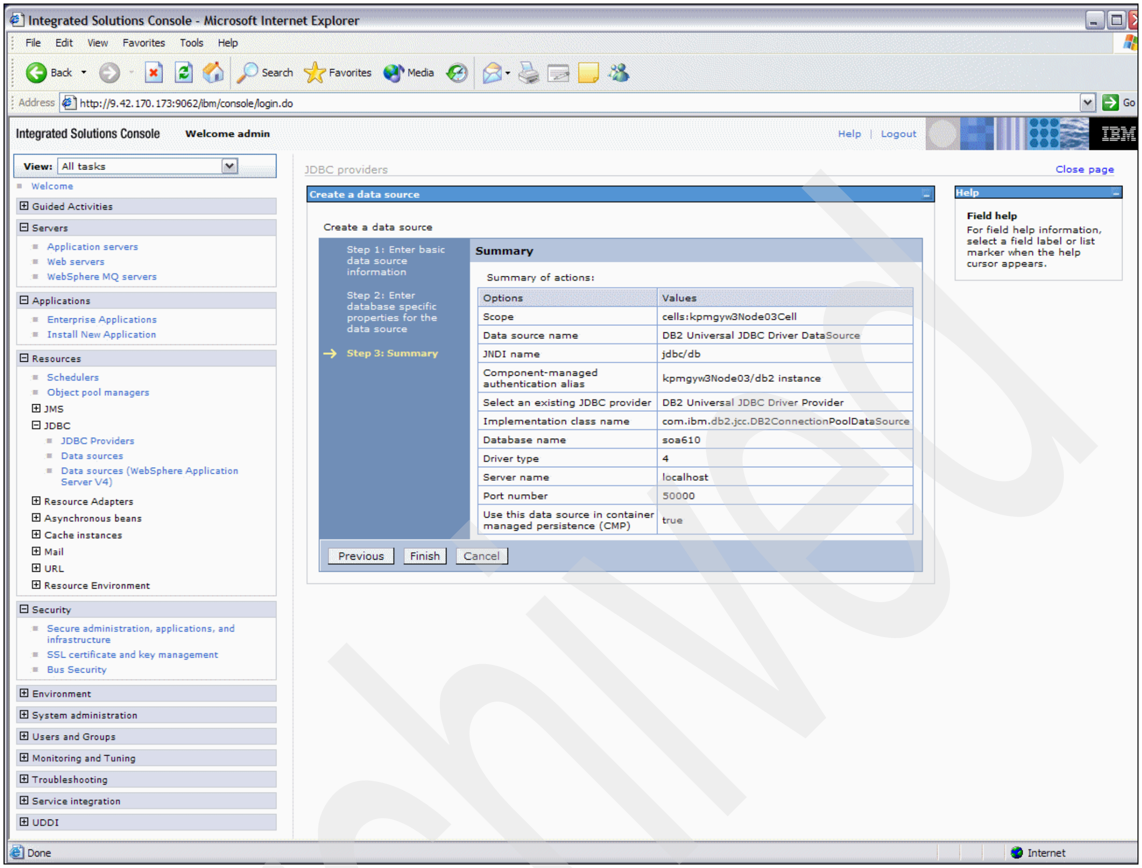


Figure B-43 Summary of data source definition

Test the connection

1. Select the associated check box for the data source.
2. Click **Test connection.**
3. You should see a window similar to Figure B-44.

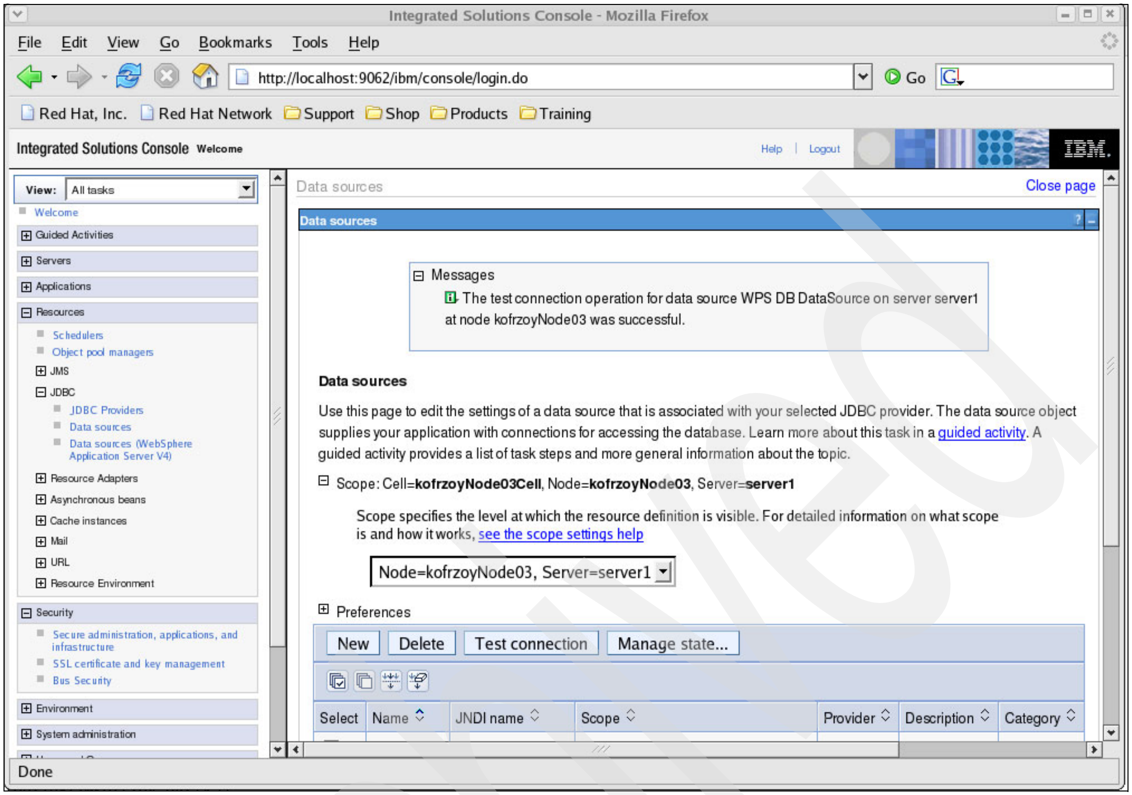


Figure B-44 Testing the connection with DB2

B.6 Deploy PS application

The IBM WebSphere Presence Server component is installed as an enterprise application in the WebSphere Application Server.

1. Log in to the WebSphere Integrated Solutions Console.
2. In the navigation panel, click **Applications** → **Install new Application.**
3. Click **Browse** to locate the `WebSpherePresenceServerEAR.ear` file on your system.
4. Click **Next** → **Next** → **Next** → **Finish.**
5. Click **Save** to save to the master configuration.

Start PS Application

1. In the WebSphere Integrated Solutions Console navigation panel, click **Applications → Enterprise Applications.**
2. Select **Presence Server** .
3. Click **Start.**
4. The PS application should be started successfully.
5. Check for any errors in the SystemOut log file.

Configure the PS application

1. Open `SystemConfiguration.xml` with a text editor.
2. Update the values for the following attributes as needed.

Table B-5 SystemConfiguration.xml attributes

Attribute	Value
enable-authorization	Set value to true to use authorization applications ► You can keep the default
usage-record	Set value to true to log usage records ► You can keep the default
subscribe-expiration	Set minimum and maximum values to control the expiration time for SUBSCRIBE requests ► You can keep the default
publish-expiration	Set minimum and maximum values to control the expiration time for PUBLISH requests ► You can keep the default
enable-xproviders	Set value to true to enable the Xproviders mechanism ► You can keep the default
glms-configuration	Set the values to identify the Group List Server address and user ► This parameter **must** contain the SIP address that is used for SUBSCRIBE requests issued by the PS, and the user ID and password that is used to retrieve XCAP documents from the GLS.
scscf-configuration	This value is used to specify the SIP address of the S-CSCF server

Attribute	Value
asserted-identity	Set the value to the URI for the identity of the requestor in the scheme:identity format ▶ This parameter MUST contains the asserted identity of the PS. The whole assert identity must be quoted, and the “Super Admin” string has to be quoted separately. Since quote between quote is not accepted, you must use the **"** expression. the < and **>** characters are not accepted, therefore you must use **<** and **>** expressions

Figure B-45 on page 623 shows values of the SystemConfiguration file for this sample application test environment.

```
root@kpmgyw3:/opt/IBM/PS/Configuration
<?xml version="1.0" encoding="UTF-8"?>
<system-configuration>

        <!--  disabling/enabling the authorization mechanism  -->
        <enable-authorization value="false" />

        <!--  Configuring generation of usage records per sip request -->
        <usage-record publish="true" subscribe="true" notify="true" />

    <!-- configuring default values (in minutes) for expiration of SIP subscribe requests -->
        <subscribe-expiration default="60" maximum="1440" minimum="1"/>

    <!-- configuring default values (in minutes) for expiration of SIP publish requests -->
        <publish-expiration default="60" maximum="1440" minimum="1"/>

    <!-- diasbling/enabling the Xproviders mechanism -->
        <enable-xproviders value="true" />

    <!-- Configuring GLMS server, the sip-address for SUBSCRIBE request, and authorized user to get XCAP documents -->
        <glms-configuration sip-address="sip:9.42.170.173:5063" user="GLSSuperAdmin" password="itso4you" />

    <!-- Configuring S-CSCF server, only the sip-address to send SUBSCRIBE requests is required -->
        <scscf-configuration sip-address="" />

    <!-- The asserted identity for the presence server -->
        <asserted-identity value="&quot;Super Admin&quot; &lt;glm:GLSSuperAdmin&gt;" />

</system-configuration>
~
```

Figure B-45   SystemConfiguration.xml file

3. Save and close the file.
4. Open `CmdConfigParams.txt` with a text editor.
5. Change the following parameters.

Table B-6   CmdConfigParams.txt parameters

Attribute	Value
cfg.system	path_to_SystemConfiguration.xml

Attribute	Value
username	database_administrator_user_name for example: `d2inst1`
password	database_administrator_password
dbConnectionString	jdbc:db2://database_host_name:database_port/database_name for example: `jdbc:db2://localhost:50000/SOA610`

Figure B-46 on page 624 shows values of the CmdConfigParams file for this sample application test environment.

```
root@kpmgyw3:/opt/IBM/PS/Configuration
#######################################################################################
#                               Presence server configuration file
#######################################################################################
#
############################## Component to be configured: ##################################
cfg.system = /opt/IBM/PS/Configuration/SystemConfiguration.xml
#
######################### User properties username, password ################################
username = db2inst1
password= itso4you
#
######################### DB connection string for jdbc usage ##################################
#for DB2 use jdbc:db2://<host name>:<port number>/<DB name>
#for Oracle use:jdbc:oracle:thin:@<host name>:<port number>:<server instance >
#
dbConnectionString= jdbc:db2://localhost:50000/SOA610
#
######################### dbDriver ######################################################
#for DB2 use: com.ibm.db2.jcc.DB2Driver
#for Oracle use: oracle.jdbc.driver.OracleDriver
#
dbDriver=com.ibm.db2.jcc.DB2Driver
#
#######################################################################################
#                       Please don't change this section
#######################################################################################
tableName= Config
keyColumn= key
valueColumn= value
#######################################################################################
~
```

Figure B-46 CmdConfigParams.txt file

6. Run the following command:

   java -classpath **WPSConfigurationUtils.jar:** path_to_all_jdbc_drivers **CmdConfig** path_to_CmdConfigParams.txt/**CmdConfigParams.txt**

Important: Enter the following parameters on a single line. JDBC drivers must be separated by a colon”**:**” character.


```
[root@kpmgyw3 Configuration]# /opt/IBM/WebSphere/AppServer/java/bin/java -classpath WPSConfigurationUtils.jar:/opt/IBM/db2/V8.1/java/db2jcc.jar:/h
ome/db2inst1/sqllib/java/db2jcc_license_cu.jar:/home/db2inst1/sqllib/java CmdConfig /images/imsbeta/IBMWebSpherePresenceServer/Server_application/
Configuration/CmdConfigParams.txt
File loader created.
Property file loaded.
Property file parsed.
There are 1 elements to be configured.
XML file name: /images/imsbeta/IBMWebSpherePresenceServer/SystemConfiguration.xml
XML source loaded from /images/imsbeta/IBMWebSpherePresenceServer/SystemConfiguration.xml
DB manager created
Connecting with user: db2inst1
Connecting to: jdbc:db2://localhost:50000/SOA610
DB connection created.
DB ResultSet created
Storing com.ibm.wkplc.presence.configuration.SystemConfiguration in progress..
DB table is empty
SQL executed.
DB connection closed
XML stored in DB.
Done.
-------------------------------------------------------------------------------
[root@kpmgyw3 Configuration]#
```

Figure B-47 Running the command

Restart the PS application server

1. To stop the server, run the following command from the application server profile:

   `was_profile_root/bin/stopServer.sh server_name`

   Where:

   `server_name` is the name of the application server (this should be `server1`)

2. To restart the server, run the following command:

   `was_profile_root/bin/startServer.sh server_name`

   Where:

   `server_name` is the name of the application server.

B.7 IBM WebSphere Diameter Enabler component

WebSphere Diameter Enabler component are deployed on the WebSphere Application Server platform. The sample application only uses the Rf accounting Web Services, so only the necessary components are deployed. For example, the support of Rf Accounting does not require any database to be configured.

Note: WebSphere Application Server Network Deployment, Version 6.1 must be installed on the server before beginning the installation of WebSphere Diameter Enabler components

The process outlined here describes the steps for the minimum installation. Additionally, it describes the steps for installing the simulator that supports basic interaction with the Diameter client. The outline of the installation processes consist of the following:

- B.7.1, “Install base binaries” on page 626
- B.7.2, “Create WebSphere Application Server profile” on page 626
- B.7.3, “Deploy the Diameter Enabler application on WebSphere Application Server” on page 627
- B.7.4, “Deploy Diameter Rf Web Services” on page 634

> **Note:** The installation process assume that Linux Red Hat Enterprise Linux AS 4.0 Update 3 is being used, and WebSphere Application Server 6.1 is installed and configured on the machine.

B.7.1 Install base binaries

1. Create **/opt/IBM/Diameter** directory.
2. Copy and/or extract all the binaries into **/opt/IBM/Diameter** directory.

B.7.2 Create WebSphere Application Server profile

An application server dedicated to the Presence Server. This removes the potential for conflicts between components for resources allowing performance tuning of the Java Virtual Machine on a product by product basis.

1. Switch to `<was_root>/firststeps` directory.
2. Start the first steps GUI `./firststeps.sh`.
3. After the wizard opens, perform the following.
   a. Select the profile management tool.
   b. Click **Next.**
   c. Select **Application Server** as the type of WebSphere Server environment to create.
   d. Click **Next.**
   e. Choose **Typical Profile Creation** as profile creation process.
   f. Deselect **Enable Administrative Security** to disable Administrative Security.
   g. Click **Next.**
   h. Review the profile creation summary.
   i. Click **Create.**

A window similar to Figure B-48 on page 627 will be displayed.

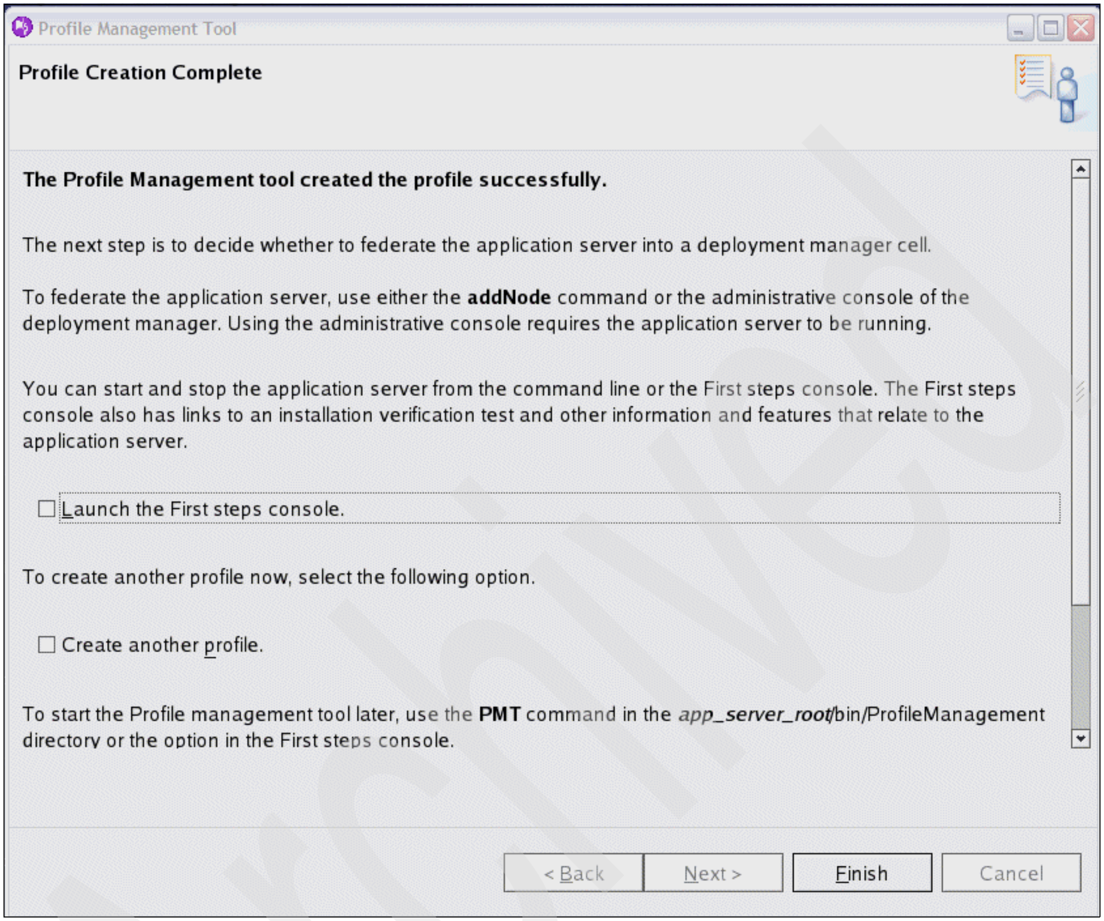


Figure B-48 Diameter profile creation in WebSphere Application Server

B.7.3 Deploy the Diameter Enabler application on WebSphere Application Server

Complete the following steps to deploy the Diameter Enabler application.

1. Copy `com.ibm.ws.diameter_6.1.0.jar` to the WebSphere Application Server plugins directory:

   ```
   was_root/plugins
   ```

2. Open a command prompt.

3. Configure the Diameter channels.

```
was_profile_root/bin/wsadmin.sh -username user_name -password
password -f script_path/DiameterChannelInstall.py cell_name
node_name server_name host_name port_number standalone [debug]
```

Where:

– user_name

  Is your WebSphere Application Server user ID

– password

  Is the password associated with your user_name

– script_path

  Is the path to `DiameterChannelInstall.py`

– cell_name

  Is the name of cell where the server is installed

– node_name

  Is the name of node where the server is installed

– server_name

  Is the name of the server

– host_name

  Is the fully qualified host name where the node is installed

– port_name

  Is the port number for the Diameter inbound TCP channel, 3868 preferred

– standalone

  Indicates that the script is running in a standalone environment

– [debug]

  Enables debugging for the configuration script

Important: You must enter the parameters on a single line.

```
[root@kofrzoy bin]# ./wsadmin.sh -f /opt/IBM/IBM_WebSphere_Diameter_Services/Configuration/DiameterChannelInstall.py kofrzoyN
ode01Cell kofrzoyNode01 server1 localhost 3868 standalone debug
WASX7209I: Connected to process "server1" on node kofrzoyNode01 using SOAP connector;  The type of process is: UnManagedProce
ss
WASX7303I: The following options are passed to the scripting environment and are available as arguments that are stored in th
e argv variable: "[kofrzoyNode01Cell, kofrzoyNode01, server1, localhost, 3868, standalone, debug]"
debug is enabled!
Parameters are:
        cell:  kofrzoyNode01Cell
        node:  kofrzoyNode01
        serverName:  server1
        host:  localhost
        port:  3868
        cluster:  0
```

Figure B-49   Diameter Channel Installation

Restart the application server

1. To stop the server, run the following command from the application server profile:

   `was_profile_root/bin/stopServer.sh server_name`

   Where:

   `server_name` is the name of the application server (this should be `server1`).

2. To restart the server, run the following command:

   `was_profile_root/bin/startServer.sh server_name`

   Where:

   `server_name` is the name of the application server.

Verifying the WebSphere Diameter channel configuration

1. Click **Servers** → **Application servers** → **server_name** → **Ports**.
2. Verify the endpoint is properly configured.
   a. Verify the `DiameterNamedEndPoint` appears in the list with either of the following values:
      i. Host: *
      ii. host_name Port: `38683`
3. Verify the `DiameterChain` is properly configured.
   a. Click **View associated transports** on the `DiameterNamedEndPoint` row.
   b. Verify `DiameterChain` appears in the list with the following values:
      i. Enabled: `Enabled`
      ii. Host: * or host_name Port: `3868`
      iii. SSL Enabled: `Disabledc`
   c. Click **OK.**

4. Verify the `SecureDiameterChain` is properly configured.

   a. Click **View** associated transports on the `DiameterNamedEndPoint` row.

   b. Verify `SecureDiameterChain` appears in the list with the following values.

      i. Enabled: `Enabled`

      ii. Host: * or host_name Port: `3868`

      iii. SSL Enabled: `Enabled`

   c. Click **OK.**

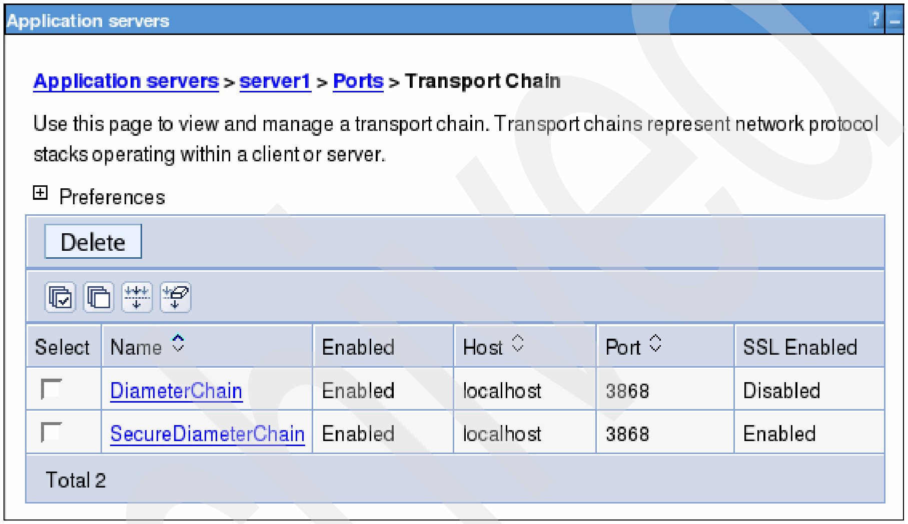


Figure B-50 Diameter Channels

5. Verify the `DiameterChain channel` is properly configured.

   a. Click `DiameterChain`.

   b. Verify the following information is correct in the Transport Channels section.

      i. TCP inbound channel: `DiameterTCPInboundChannel`

      ii. Host: * or <server_name>

      iii. Port: `3868`

      iv. Thread pool: `DiameterThreadPool`

      v. Generic inbound channel: `DiameterGenericInboundChannel`

   c. Click `Generic inbound channel (DiameterGenericInboundChannel)`.

   d. Verify the following values are correct.

      i. Transport Channel Name: `DiameterGenericInboundChannel`

ii. Discrimination weight: `1`

iii. JAR file: `com.ibm.ws.diameter_6.1.0.jar`

iv. Channel type identifier: `DiameterInboundChannel`

v. Configuration URI: this field should be empty

e. Click **OK** on the DiameterChain panel.

f. Click **OK** to return to the Transport chain panel.

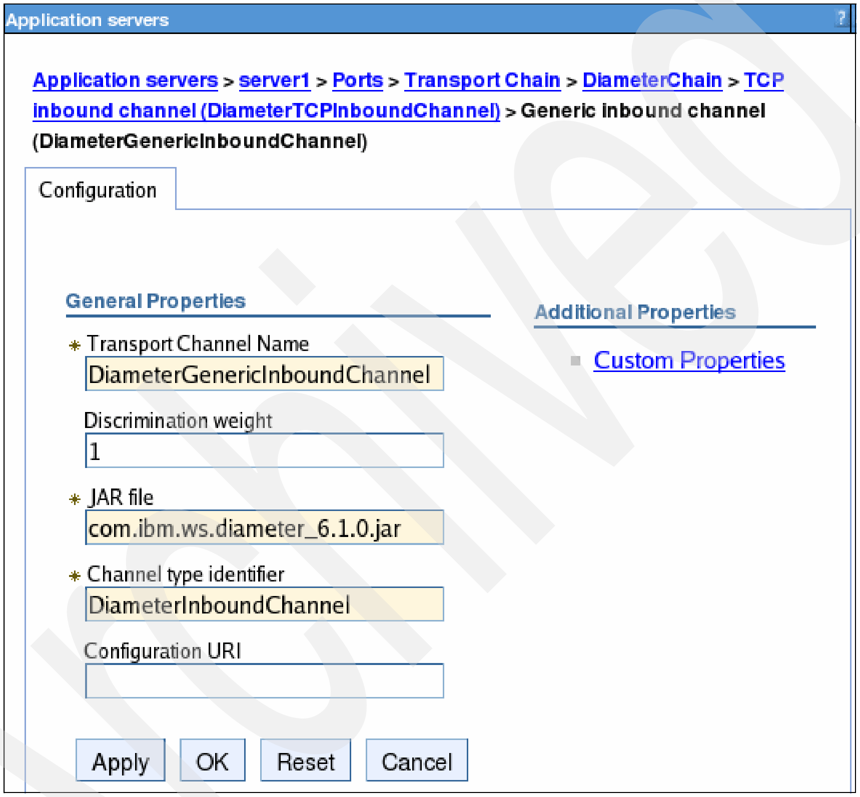


Figure B-51 Diameter Generic inbound channel

6. Verify the `SecureDiameterChain channel` is properly configured.

a. Click `SecureDiameterChain`.

b. Verify the following information is correct in the Transport Channels section.

i. TCP inbound channel: `DiameterTCPInboundChannel`

ii. Host: * or <server_name>

iii. Port: `3868`

iv. Thread pool: `DiameterThreadPool`

v. SSL inbound channel: `DiameterSSLInboundChannel`

vi. SSL Configuration: `Diameter`

vii. Generic inbound channel: `DiameterGenericInboundChanne`

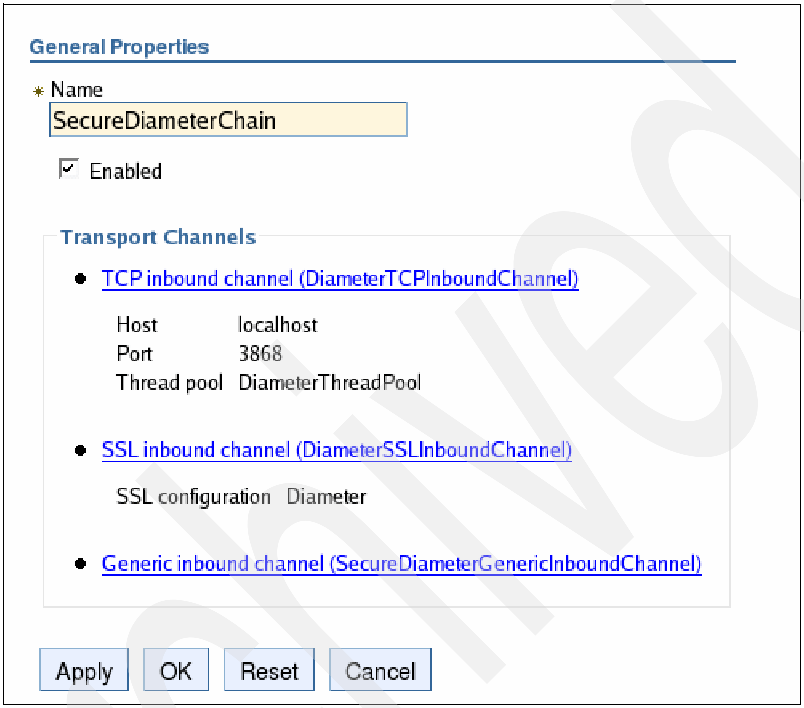


Figure B-52 Diameter Secure Chain Channel

c. Click `Generic inbound channel (DiameterGenericInboundChannel)`.

d. Verify the following values are correct.

i. Transport Channel Name: `SecureDiameterGenericInboundChannel`

ii. Discrimination weight: `10l`

iii. JAR file: `com.ibm.ws.diameter_6.1.0.jar`

iv. Channel type identifier: `DiameterInboundChannel Configuration URI should be empty`

v. Click **OK** on the SecureDiameterChain panel.

e. Click **OK** to return to the Transport chain panel.

7. Verify the SSL configuration object has been created and configured properly.

a. In the navigation panel, click **Security** → **SSL security and key management** → **SSL configurations**.

b. Click Diameter in the list of SSL configurations.

c. Under Additional Properties, click Quality of protection (QoP) settings.

d. Verify the following values are correct.

i. Client authentication: `Required`

ii. Protocol: SSL_TLSe.

e. Click **OK.**

Install Rf accounting Web Service

The Rf accounting Web Services is a messaging interface to enable an application to send accounting messages to a Charging Collection Function (CCF)

> **Note:** The CCF is simulated in this sample application scenario

1. Copy `Diameter_Rf.properties` to the `was_root/lib/` directory.
2. Open `Diameter_Rf.properties` in a text editor.
3. Find the `OriginHostName` property.

   Enter the matching host name of the application server where the WebSphere Diameter Enabler base is installed.
4. Find the `OriginRealmName` property.

   Enter the matching realm name of the application server where the WebSphere Diameter Enabler base is installed.
5. Find the `HostIpAddress` property.

   Enter the IP address where the WebSphere Diameter Enabler base is installed.
6. Find the `ProxySupport` property.

   Enter `false` to turn off proxy support.
7. A connection with a remote peer needs to be defined. For the purpose of the sample application the remote peer is the CCF simulator. To configure a connection with a remote peer do the following:

   a. Find the `con1.remotePeerOriginHostName` property.

   Enter the matching host name where the CCF simulator is or will be installed. It can be `OriginHostName`.

b. Find the `con1.remotePeerIPAddress` property.

   Enter the IP address of the host name where the CCF simulator is or will be installed.

c. Find the `con1.remotePeerPort` property.

   Type the port number that is used by the CCF simulator for receiving the Diameter packets. This port number must correspond to the port number that will be configured in the CCF simulator to listen for incoming Diameter packets.

d. Keep the other parameters unchanged.

```
# The OriginHostName is the unique name for this Diameter Client.
# The OriginHostName can be the same as the DNS hostname provided
# that there is only one Diameter Client executing on a given host.
# The OriginHostName will be used to identify this Diameter Node
# to those that it shares connections with.  The OriginHostName is
# required and must be a qualified domain name.
OriginHostName = kpmgyw3.itso.ral.ibm.com

# The OriginRealmName is the of which this Diameter Client resides.
# Because this is a Diameter Client, the OriginRealmName is only used
# as information exchanged in the Capabilities Exchange (CER/CEA) transaction.
# The OriginRealmName is required and must be a qualified domain name.
OriginRealmName = itso.ral.ibm.com

# The HostIpAddress is the IP Address of this Diameter Client. If the Diameter
# Client is running on a multi-honed machine, the HostIpAddress should represent
# the network interface that the Application Server is using. This value is only
# used as information exchanged in the Capabilities Exchange (CER/CEA) transaction.
# The HostIpAddress is required and the value will be verified against the valid IP
# Addresses for this host machine.
HostIpAddress = 9.42.170.173

# ProxySupport is a boolean value that indicates whether this Diameter Client will
# allow requests made from it to potentially be passed through a proxy agent
# in the Diameter network. If set to false, no packets originating from the Diameter
# client will have the P bit set. If set to true, packets may have the P bit set
# depending on the command code in use. The default value is true.
ProxySupport = false
```

Figure B-53 Diameter Rf Web Service configuration

B.7.4 Deploy Diameter Rf Web Services

1. Log in to the Integrated Solutions Console.
2. In the navigation panel, click **Applications** → **Install new Application.**
3. Click **Browse** to locate the `DHADiameterRfWebServiceEAR.ear` file on your system.

4. Click **Next** → **Next** → **Next** → **Finish.**
5. Click **Save** to save the master configuration.

Start the application server

1. In the navigation panel, click **Applications** → **Enterprise Applications.**
2. Select `DHADiameterRfWebServiceEAR`.
3. Click **Start.**
4. The `DHADiameterRfWebServiceEAR` application should be started successfully.
5. Check SystemOut log file to for any errors.

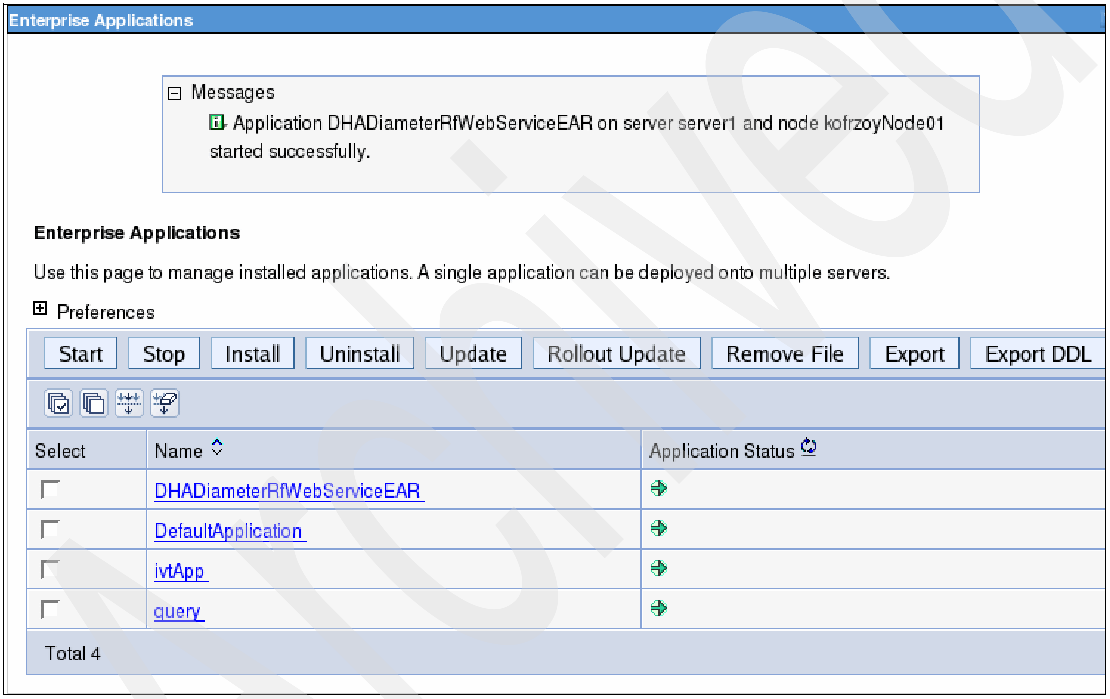


Figure B-54 DHADiameterRfWebServiceEAR application start

C

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

`ftp://www.redbooks.ibm.com/redbooks/SG247255`

Alternatively, you can go to the IBM Redbooks Web site at:

`ibm.com/redbooks`

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG247255.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name *Description*

7255code.zip Zip file containing code samples and the SipXPhone softphone.

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 40 GB Disk
Operating System: Windows or Linux
Processor: Minimum 1GHz
Memory: Minimum 1 GB

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Description of sample code

Table 13-3 describes the contents of the 7255code.zip file after unzipping.

Table 13-3 Contents of 7255code.zip

Item	Description
SIP Sample applications	Code for the Registrar and proxy, and Third Party Call Control SIP applications from chapters 9 and 10.
IMS Sample applications, FindHelp_RSAspec_062306.zip	Code for the FindHelp IMS application from chapters 11, 12 and 13.
sipXphone-2_6_0_27.exe	sipXphone softphone executable file

Abbreviations and acronyms

3GPP	Third Generation Partnership Project
3GPP2	Third Generation Partnership Project 2
3PCC	Third Party Call Control
AAA	Authentication, Authorization and Accounting
AIN	Advanced Intelligent Network
AIP	Application Infrastructure Provider
ARIB	Association of Radio Industries and Businesses
AS	Application Server
ATIS	Alliance for Telecommunications Industry Solutions
AUID	Application Unique ID
B2BUA	Back-To-Back User Agent
BGCF	Breakout gateway Controller Functions
BPEL	Business Process Execution Language
CAMEL	Customized Applications for Mobile Networks Enhanced Logic
CBE	Common Base Event
CCF	Charging Collection Function
CCSA	China Communications. Standards Association
CDR	Call Detail Record
CEI	Common Event Infrastructure
CPE	Customer Premise Equipment
CS	Circuit switched
CSCF	Call Session Control Function
CSCF	Call Session Control Function
Cx Reference Point	Diameter based interface to HSS
DB2	IBM universal Data Base 2
DMZ	DeMilitarized Zone
EAR	Enterprise ARchive
EJB	Enterprise JavaBean
ESB	Enterprise Service Bus
ETSI	European Telecommunication Standards Institute
ETSI	European Telecommunications Standards Institute
ETSI	European Telecommunications Standards Institute
FMC	Fixed Mobile Convergence
GLMS	Group List Management Server
GLS	Group List Server
GUP	3GPP Generic User Profile
HSS	Home Subscriber Server
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
ICMP	Internet Control Management Protocol
I-CSCF	Interrogating Call Session Control Function
IDE	Integrated Development Environment
IMS	IP Multimedia Subsystem
IMS	IP Multimedia Subsystem
IMS GWF	IMS Gateway Function
IM-SSF	IP Multimedia Service Switching Function

IMT-2000	International Mobile Telecommunications-2000
IPSEC	Internet Protocol Security
IPTV	Internet Protocol Television or Interactive Programming Television
ISC	IMS Service Control
ISG	Intelligent Services Gateway
ITSO	International Technical Support Organization
ITU	International Telecommunications Union
IVR	Interactive Voice Response
J2EE	Java 2 Platform, Enterprise Edition
JAAS	Java Authentication and Authorization Service
JAIN	Java Advanced Intelligent Network
JMS	Java Message Service
JMX	Java Management eXtensions
JVM™	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
LNP	Local Number Portability
MDS	Messaging and Data Services
MGCF	Media Gateway Control Function
MGW	Media Gateway
MRF	Media Resource Function
MRFC	Multimedia Resource Function Controller
MRFP	Media Resource Function Processor
MVNE	Mobile Virtual Network Enabler
MVNO	Mobile Virtual Network Operator
NAS	Network Access Server
NGN	Next Generation Network
NGN	Next Generation Network
NGOSS	Next Generation Operational Support System
NLS	Natural Language Support
OCF	Online Charging Function
OCS	Online Charging System
OSA	Open Services Architecture
OSA-SCS	Open Service Access - Service Capability Server
OSS/BSS	Operational Support Systems/Business Support Systems
P-CSCF	Proxy Call Session Control Function
PS	Presence Server
PSTN	Public Switched Telephone Network
PSTN	Public Switched Telephone Network
PTT	Push to Talk
QoS	Quality of Service
Quad Play	Integrated voice, video, data and mobility offering
RADIUS	Remote Authentication Dial In User Service
RAF	Repository Access Function
RTCP	Real-time Transport Control Protocol
RTP	Real-time Transport Protocol
SAR	SIP Application Resource
SCE	Service Creation Environment
SCF	Session Control Function
SCIM	Service Capability Interaction Manager
SCIP	Simple Conference Invitation Protocol

S-CSCF	Serving Call Session Control Function
S-CSCF	Serving Call Session Control Function
SCTP	Session Control Transmission Protocol
SDK	Software Development Kit
SDO	Service Data Object
SDP	Session Description Protocol, or, Service Delivery Platform
SGW	Signaling GateWay
SID	Shared Information & Data
SIMPLE	SIP for Instant Messaging & Presence Leveraging Extensions
SIP	Session Initiation Protocol
SIP AS	Session Initiation Protocol Application Server
SIPPING	Session Initiation Protocol Project INvestiGation
SIPS	Secure SIP
SLEE	Service Logic Execution Environment
SLF	Subscription Locator Function
SMTP	Simple Mail Transmission Protocol
SNMP	Simple Network Management Protocol
SOA	Service-oriented architecture
SOAP	Simple Object Access Protocol
SPI	Service Provider Interface
SSL	Secure Sockets Layer
SWG	SoftWare Group
TCAP	Transmission Capabilities Application Protocol
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
TIA	Telecommunications Industry Association
TISPAN	Telecoms & Internet converged Services & Protocols for Advanced Networks
Triple Play	Integrated Voice, Video & Data Offering
TTA	Telecommunications Technology Association
TTC	Telecommunication Technology Committee
TTS	Text-to-Speech
TUI	Telephony User Interface
UA	User Agent
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
VoD	Video on Demand
VoIP	Voice over IP
VPN	Virtual Private Network
VXML	Voice Extensible Markup Language
WAR	Web ARchive
WAS	WebSphere Application Server
WID	WebSphere Integration Developer
WiMAX	Worldwide Interoperability for Microwave Access
WSDL	Web Services Description Language
XCAP	XML Configuration Access Protocol
XCAP	XML Configuration Access Protocol
XML	eXtensible Markup Language

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 645. Note that some of the documents referenced here may be available in softcopy only.

- *Technical Overview of WebSphere Process Server and WebSphere Integration Developer*, REDP-4041

  `http://www.redbooks.ibm.com/abstracts/redp4041.html?Open`
- *Patterns: Building Serial and Parallel Processes for IBM WebSphere Process Server V6*, SG24-7205

  `http://www.redbooks.ibm.com/abstracts/sg247205.html?Open`
- *WebSphere Application Server V6.1: Technical Overview*, REDP-4191

  `http://www.redbooks.ibm.com/abstracts/redp4191.html?Open`
- *WebSphere Application Server V6.1: System Management and Configuration*, SG24-7304

  `http://www.redbooks.ibm.com/abstracts/sg247304.html?Open`
- *Rational Application Developer V6 Programming Guide*, SG24-6449

  `http://www.redbooks.ibm.com/abstracts/sg246449.html?Open`
- *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303

  `http://www.redbooks.ibm.com/abstracts/sg246303.html`
- *Patterns: Model-Driven Development Using IBM Rational Software Architect*, SG24-7105

  `http://www.redbooks.ibm.com/abstracts/sg247105.html`
- *Getting Started with WebSphere Enterprise Service Bus V6*, SG24-7212

  `http://www.redbooks.ibm.com/abstracts/sg247212.html?Open`
- *Enabling SOA Using WebSphere Messaging*, SG24-7163

  `http://www.redbooks.ibm.com/abstracts/sg247163.html`

Other publications

These publications are also relevant as further information sources:

- *SIP: Understanding the Session Initiation Protocol,* by Alan B. Johnston. Artech House Publishers; Second edition (November 2003), ISBN-10: 1580536557
- *The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds,* by Gonzalo Camarillo and Miguel A.García-Martín. Wiley; Second edition (February 2006), ISBN-10: 0470018186

Online resources

These Web sites and URLs are also relevant as further information sources:

- Business Process Execution Language for Web Services Version 1.1

  http://www-128.ibm.com/developerworks/library/specification/ws-bpel/
- Service Component Architecture

  http://www-128.ibm.com/developerworks/webservices/library/specification/ws-sca/
- Service Data Objects

  http://www-128.ibm.com/developerworks/webservices/library/specification/ws-sdo/
- IBM WebSphere Developer Technical Journal: Session Initiation Protocol in WebSphere Application Server V6.1 - Part 1

  http://www-128.ibm.com/developerworks/websphere/techjournal/0606_burckart/0606_burckart.html
- IBM WebSphere Developer Technical Journal: Session Initiation Protocol in WebSphere Application Server V6.1 - Part 2

  http://www-128.ibm.com/developerworks/websphere/techjournal/0608_burckart/0608_burckart.html
- Tuning IBM WebSphere Telecom Web Services Server

  http://www-1.ibm.com/support/docview.wss?uid=swg27008225&aid=1
- Service-oriented modeling and architecture

  http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/
- Elements of Service-Oriented Analysis and Design

  http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/

- Building SOA applications with reusable assets: Reusable assets, recipes, and patterns

  http://www-128.ibm.com/developerworks/webservices/library/ws-soa-reuse1/
- Reusable Asset Specification Repository for Workgroups

  http://www.alphaworks.ibm.com/tech/rasr4w
- UML basics: An introduction to the Unified Modeling Language

  http://www-128.ibm.com/developerworks/rational/library/769.html
- Rational UML Profile for business modeling

  http://www-128.ibm.com/developerworks/rational/library/5167.html
- Introducing IBM Rational Software Architect

  http://www-128.ibm.com/developerworks/rational/library/05/524_rsa/

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Numerics

3G 28–29
3GPP 28–29, 31, 34–36, 160, 173–174, 177, 303
3GPP2 28, 174–175
3rd Generation Partnership Project 28, 173
See also 3GPP
3rd Generation Partnership Project 2 28, 173
See also 3GPP2

A

AAA 35–36, 178
See also Authentication, Authorization, and Accounting
ACK 11–15, 18–19, 24, 209, 217, 224, 275, 296, 325, 474, 476
AddressesBO 338, 364–368, 371, 428
xsd 428
AddressOfRecord 243–245
Admission Control 199, 584–585
AdmissionControlInterface 584
Alliance for Telecommunications Industry Solutions 28
A-Party 314, 316, 319, 323–325, 337
APIs 20, 59, 83, 86, 117, 170
See also Application programming interfaces
AppConfigHTTP 83–84
Application
business logic 200
code 238
components 131, 229
composition 208, 221, 229–230, 234, 257
deployment 64, 450
development 54, 63–65, 120, 499, 503
development environment 63–64, 120, 499
development tool 64
environment 53
features 5
functions 67
installation 449, 570, 572–573, 575, 577–578, 580
instances 25
integration 486
interoperability 20
platforms 160
portability 165, 167
profile 541
status 607
support 167
types 50
usage 190
Application programming interfaces 20, 181
See also APIs
Application Server 20, 31–34, 36–37, 39, 58–59, 100, 174, 176–177, 181, 231, 568–569, 629
instance 280, 600
node 448, 458, 481–482, 484, 486
profile 495, 536, 601, 617, 625, 629
Application Server Toolkit 50–51, 53, 55, 63–64, 89, 166, 233–234, 236, 258, 293, 319, 342–343, 455, 500–501, 582
Develop using 258
enhancements 166
install 236, 500
installation wizard 500
See also AST
AppServer 481–482, 484, 495–496, 525, 534, 545, 554
appSession 214, 225, 228, 263–264, 269, 274–278
getAttribute 228, 263–264, 269
setAttribute 274, 277–278
ArrayList 352, 357–359
availableGroupMembers 359
ArrayOf 379, 382
Association of Radio Industries and Businesses 28
AST 53–54, 63–66, 69, 72, 82–83, 88–90, 98, 100, 105, 166, 234, 236, 239, 247, 256, 258, 289, 343, 352, 455, 500, 502, 508–509, 513–514
Install 500
See also Application Server Toolkit
Starting 502
AUIDs 192
Authentication
alias 561–563, 565, 575, 594, 596, 618–619
create 561–562, 594, 618
challenge 226
Component-managed 565, 596, 619
configuration 282

type 226
Authentication, Authorization, and Accounting 35
See also AAA
Authorization
API 59, 100
applications 622
constraints 79, 226
Service 562, 594, 618

B

B2BUA 8, 33, 37, 59, 101, 175
See also Back-to-Back User Agent
Back-to-Back User Agent 8, 37, 59, 175
See also B2BUA
BGCF 33–34
See also Breakout Gateway Controller Functions
BPEL
Engine 304, 440
flow 128, 307–309, 450–451, 477, 479, 483–484
MyService 131
process 57, 123–124, 131–133, 309, 319–320, 324–325, 328, 336–337, 341–342, 348, 361, 368, 393, 418, 430, 435, 440
runtime environment 142
See also Business Process Execution Language
Web service 358
Breakout Gateway Controller Functions 33
See also BGCF
Business
applications 200
integration 120, 122–123, 126–128, 135, 142, 361, 363–364, 366, 383, 387, 390, 417, 431–432, 450–451, 477
modules 123
perspective 126–127, 363–364, 431–432, 450, 477
logic 127, 176, 200, 229, 238, 242, 244, 246, 262, 270, 342, 361, 383, 390
objects 121, 128, 133, 200, 364, 371
processes 52, 60, 120, 122, 134–135
rules 120–121, 124, 132, 201
Business Process Execution Language
See also BPEL
BYE 11–12, 19, 24, 208–209, 272–276, 278, 296, 325, 334, 360, 474–475, 477
message example 334
Request 19, 272–273, 275, 360

C

Call Control
Servlet 276
SipServlet 270
create 270
Call Controller 215, 275–277
call flow 7, 17–19, 102, 328–329
Call Forwarding 9, 83, 85–87
Call Session Control Function 36, 38, 59
See also CSCF
CallControl 227, 259–263, 265, 268, 271–273, 287
EAR 279
SIP Servlet 259, 261–263, 273
SipServlet 272
create 272
CallController 215, 221, 264, 274, 277, 289–290, 293
doSuccessResponse 274
servlet 221, 277
thirdPCC 264
CallControlStatusRPC 268
create 268
Call-ID 11, 16, 211, 216, 231, 244, 296, 330–331, 334, 470, 475–477
CAMEL 34, 37
Application part 37
CANCEL 8, 11–12, 24, 208–209, 215, 217, 264, 296
CCF 179, 181, 322, 325, 441, 445, 465–466, 486, 633–634
See also Charging Collection Function
Simulator 322, 325, 445, 465–466, 486, 633–634
CDATA 470, 475–477
CDMA2000 28
CEI 135–136, 149, 200
See also Common Event Infrastructure
CEI event emitter mediation primitive 149
Charging
event 319, 390, 392, 411–413, 416, 486
create 390, 392, 411–413, 416
services 361, 373
Charging Collection Function 181–182, 322, 465, 633
See also CCF
China Communications. Standards Association 28

ClickToCall 260, 266, 268
Servlet 260
create 260
Collaborations 318–322
command line interface 187, 446–447, 586, 609–611
Common Components 198
Common Event Infrastructure 135, 200
See also CEI
Common Open Policy Service 35
composite applications 49, 55, 57
Composite services 57–58, 61, 122, 301–303, 305
architecture 302
choreography 303
designing 305
IMS 302
orchestration 305
conferencing services 305
Configure
authentication 282
DB2 540, 546, 586, 589, 611–612
Service Integration Bus 554
Content
ID 331–332
Length 297, 330–331, 334, 470, 475–477
Type 297, 330–332, 334, 470, 475, 610–611
contentBody 355, 481
contentType 355, 358
Control plane 36–38, 160, 165, 174, 177
Converged
application 51, 72, 209, 227, 234, 256–257, 277, 294
container 51, 126, 169, 316
HTTP 72, 162, 167, 170, 440
Project 67, 69–71, 258, 342–343
services 29, 39, 162
servlet 167, 209, 227, 268
SIP 49, 65–66, 69, 71, 163, 169, 233, 258, 280, 307
SIP/HTTP 65, 69, 71–72, 162–163, 170, 233, 440
ConvergedServlet 227–228, 260–261, 268–269
example 227
COPS 35
createRequest 213–215, 275–276, 278, 356, 360
CSCF 32–34, 36, 59, 101, 164, 174–177, 197, 302, 305, 323, 474, 622
Interrogating 32
Proxy 32
See also Call Session Control Function
Serving 32, 34, 175
CSeq 11, 16, 211, 231, 244, 296, 330–331, 334, 470, 475–477
Cx 34

D

Data
objects 121, 200
source 60, 156, 563, 565–567, 569, 594–598, 618–620
source class 597, 619
type 338–340, 364, 371–373, 393, 398, 400, 404, 407, 409–410, 412, 414
Type Selection 371–372, 398
types 128, 182, 364, 378–382
Database
configuration 547–554
name 547, 565, 589, 596, 612, 614, 619
Remote Server 547
server 547, 589, 612–613
connection port 547, 589, 612
type 563, 570, 572–573, 575, 577, 579–580, 595, 618
DB2 182, 185, 540–541, 546–547, 561–565, 570–577, 579–580, 586, 589–591, 595–596, 598–599, 611–615, 618–619, 621, 624
See also DB2 server
DB2 server 590, 613
See also DB2
Decision tables 134–135, 201
Deployment
descriptor 65–66, 68–69, 71–73, 75–76, 78, 80–81, 166, 207, 209, 214–215, 218–219, 221, 223–226, 228, 231, 238, 240–241, 247–249, 260, 266–268, 272, 277, 346–347, 349–350, 353, 456–457, 482
descriptor information 68, 72
deployment descriptor 68, 277
DHADiameterRfTestClient.java 112
DHADiameterRfWebService
EAR 381, 634–635
WAR 381
Diameter
client samples 110
Enabler 180–181, 340, 380–381, 625–627, 633
messaging interface 58, 100
Offline charging 321, 373, 380–381

Resources 58, 90
Rf 97, 101, 110–112, 321, 426, 440, 445, 486, 496, 634
Rf Service 426
RF test client 110–112
Rf Web Service 321, 486, 634
Servers 179
Sh test client 110, 118
Traffic port 444
Diameter Rf Web Services
deploy 634
DiameterGenericInboundChannel 630, 632
DiameterRfImport 426, 454
component 426
componnet 426
properties 454
DiameterRfPartner 390, 414, 427
DiameterRfService 98–99, 110, 319, 321, 325, 340, 381, 383, 390, 414, 426–427
component 321, 325
wsdl 98–99, 110, 381, 383, 427
DiameterShNotifyService 98
wsdl 98
DiameterShService 98, 110
wsdl 98, 110
Document Selector 190–191
doRequest 23, 25, 107, 210, 217
DOS command window 471–472, 485
Dx 34
Dynamic Web Projects 265–266

E

EAR 66, 84, 280, 283, 379, 381, 448, 453, 545, 570–578, 580–581, 583–586, 604–605, 621, 634
Eclipse Integrated Development Environment 45
Eclipse Modeling Framework 46
Editor
business object mapping 135–136
business process 130, 135–136, 386–387, 390, 393–396, 425
business rule group 136
business rule set 135
business state machine 135–136
decision table 135
human task 136
interface mapping 136
mediation flow 135, 147, 150, 153–154
selector 136
visual snippet 129, 135, 393, 403
EJB 54, 65, 82, 569, 571–572, 574–575, 577, 579–580
See also Enterprise JavaBeans
Enablement Toolkit 50–51, 55, 58, 89–90, 97, 100, 110, 508–509, 513
Enablers 39, 55–57, 61, 125, 131–132, 162–165, 182, 302–304, 313, 338, 361
Enterprise JavaBeans 54, 65
See also EJB
Enterprise Service Bus 52, 60–61, 120, 122, 142, 147–148, 165, 170, 194, 199–201, 304–305, 310, 312, 518, 527–528
See also ESB
ESB 146, 150–151, 153, 193–194, 199–200, 304, 312, 524, 527, 529, 545, 548–550
See also Enterprise Service Bus
Establish Call 390, 392, 407–408, 411
Ethereal 53, 482, 484–485, 488–489, 491
European Telecommunications Standards Institute 28–29
Event
charging 319, 340, 390, 392, 411–413, 416, 486
list 330–331, 356
monitor 135
exceptions 75, 130, 309
execution
environments 52, 159, 161
Exports 122, 336

F

Faults and Alarms 199
FindHelp
application 312, 314–316, 322, 344, 360–361, 460, 474
BPEL 324–325, 337, 440, 450–451, 479, 483–484
BPEL flow 450–451, 479, 483–484
BPEL process 324–325, 337, 440
caller requests script 478
component 326–328, 418, 420, 422–427, 433
interface 364
module 338, 363–364, 376, 378, 380–381, 416–417, 430, 432
process 338–339, 390, 427
process component 427
sample IMS application 361, 439–440, 466

sample service 313, 450
SAR 361, 455, 457–458
service 313, 316, 318, 323–325, 337, 341–344, 427, 440, 447, 466–467, 473–474, 480
service administrator 467
SIP Servlet 336, 352, 430, 440, 444, 455, 458–459, 478, 481–484, 496
WSDL files 427, 430
XML 474, 478, 485
FindHelpBP 383, 386–387, 390, 419
FindHelpBpelProcess 319, 321–322
component 319
FindHelpDialog 318–319
component 318
FindHelpInterface 333, 338, 369, 373, 384, 418, 421, 425, 428
WDSL 428
FindHelpInterfaceHttp 428
FindHelpInterfaceProxy 336, 353, 358–359
BPEL 359
object 358
FindHelpSipServlet 319–320
component 319
Flows
BPEL 307, 442
call 53
mediation 60, 120, 134–135, 142, 147, 150–151, 154, 157, 194, 524, 527, 529
process 146
formats 29, 182, 190
foundation services 49, 56–57, 125, 133, 302, 305
multiple 302
orchestrate 49
framework 23, 25, 35, 46, 126, 172, 229, 231, 302, 432
fromSipURI 263–265, 277
Functional Components 30–31, 33, 36
Functional planes 36–37, 160

G

Gateway Control Protocol 35
See also GCP
Gateways 31, 33, 36, 38, 56, 197
GB 88, 143, 442–443
GCP 35
See also Gateway Control Protocol
Generated Callcontrol Servlet 262
generateInfo 357–360, 484
getAddressHeader 211, 243
getAttribute 213, 222, 227–228, 263–265, 269, 274–277, 356, 358, 360
getCallInformationResponse 380
getCanonicalName 358
getLocationForGroup 436
getServletContext 213, 227–228, 264, 277, 354
getURI 218, 244–245, 247, 274, 358
getUserFromAddress 274, 276
GLMAdmin 467, 607–608
GLMConfigurator 604, 606
GLMGroupACLsCache 602
GLMGroupCache 602
Global failure 14, 216
GLS
application 601, 603, 606, 608
application server 601
deploy application 586
See also Group List Server
Self Care portlet 609
Server 446–448, 604
GLSUsers 593, 605
Gm 34
Group
mappings 605
name 354, 467, 593
Resolution 149
URIs 149
Group List Management
Self Care 607–610
Server 189, 324, 447
Service Enabler 187, 189
Group List Server
See also GLS
Self Care 606
Group Resolution 149
Group Resolution mediation primitive 149
GroupListMgr 604–606
GroupListServer 587, 590
directory 587
groupName 355
groups
link 571–572, 576–577, 579–580
GSM 28

H

H.248 35
headers 5, 10, 176–177, 183, 210–211, 213, 216,

355
Home Subscriber Server 32, 36, 59, 181–182
See also HSS
host
name 192–193, 347, 446, 448, 457, 466, 478, 541, 545, 547–553, 555, 589, 608, 612
qualified name 548–553
service logic 160
HSS 32–34, 36, 59, 164, 179–182
See also Home Subscriber Server
HTTP
applications 65, 69, 72
component 168
container 167, 169, 307
converged applications 51
interface 227
message headers 5
messages 320, 482
proxies 172
requests 209
response 207–208
See also Hypertext Transfer Protocol
server 5
services 5
Servlet container 208
Servlet filters 208
Servlets 23–24, 49, 51, 66, 70–71, 167, 169, 207–208, 343, 353
Traffic port 444
user interface 222
HttpServlet 207–208, 227–228
HttpServletRequest 23, 207–208, 227–228, 263, 269
HttpServletResponse 23, 207–208, 227, 263, 269
classes 23
object 207
HttpSession 208, 223, 263, 268–269
Human task 120–124, 132, 135–136, 201, 309, 311
Hypertext Transfer Protocol 5, 69
See also HTTP

I
IBM
Installation factory 166
IP Multimedia Subsystem 161, 541
IP Multimedia Subsystem components 541
Rational Software Development Platform 141
Telecom Web Services Toolkit 50–51, 57, 60, 119
Tivoli Directory Server 185, 586, 591
Unified Service Creation Environment 45, 161
WebSphere Group List Server 56, 182, 185–187, 316, 320, 440, 586, 593
WebSphere Integration Developer 57, 59–60, 119–120, 141, 145, 430, 516, 522
WebSphere Presence Server 56, 61, 164, 182, 320, 440, 443, 611, 621
IBMApplicationSession 269
I-CSCF 32, 34
IETF 4, 26, 35, 58, 170–171, 173–174, 178, 182–183, 189–190, 331–332, 446–447, 470
implementation
component 133
integrated 167
overview 342
import
component 125–126, 424, 433
FindHelp 353, 451
icon 423, 425–426
interfaces 338
Location 373
SAR file 456
Third Party Call Control 379
Web Services 124
wizards 71
IMS 49, 55, 57
Application Server 58–59, 100, 176, 181
applications 43, 49–51, 55, 58, 120, 130, 133, 173
architecture 30, 33, 35–36, 38, 44, 51, 178, 182, 302
components 36, 49, 179, 196, 440–441
composite service architecture 302
composite services 302
control plane 174, 177
designing services 301
developing foundation applications 90
Enablement Toolkit 50–51, 55, 58, 89–90, 97, 100, 110, 508–509, 513
enablement toolkit 55, 89
enablers 55–56, 131–132, 313
foundation applications 49, 51, 89–90, 97, 101
foundation services 56
integrated services environment 38
network 32–33, 36, 160
resource importation 93–94
resources 90, 93–97, 180

sample application scenarios 502
sample applications 53, 499, 507, 522
sample applications development 507
sample service 341
service architecture 130, 312
service components 130
service composition 303
service composition architecture 303
service creation
service creation 39, 43, 51, 57
service creation environment 51, 57
Service delivery environment 161
service execution environment 61, 164
service plane 174, 176, 312
service plane architecture 312
service plane solutions 312
services 30, 34, 38–40, 160, 163, 165, 301, 311
services architecture 38–40
services plane 165
solution 159–160, 163, 165, 301, 305
solutions 165, 182
standards-compliant SIP application 56, 173
Third Party Call 379, 582–586
Third Party Call EAR 379, 583–584, 586
WTP Tools 511, 514–515
IMS Service Control 49, 58, 101, 164, 174, 178
See also ISC
IMSServices 548, 552–553
binaries 552
IM-SSF 37
IncomingMessageEvent 296
IncomingSipServletResponse 294
INFO 12, 22, 24, 101, 126, 170–171, 177, 210, 225, 243–246, 263, 270, 274–278, 294, 296, 324–325, 334, 354–360, 474, 476, 484, 487, 491
information
callers 209
module 369
summary 502, 571, 573–574, 576, 578–579, 581
Init
method 263, 277, 353
Process 390–391, 393
initial requests 213, 227
initialization 72, 347–348
install
base binaries 544, 586–587, 611, 626
directory 525, 532
image 141, 532
new application 250
WebSphere Application Server 532
installation
directory 252, 516–517, 523, 525–526, 529, 535, 545–546
folder name 514
options 250, 449, 459, 519, 570, 572–573, 575, 577–580
options page 570, 572, 575, 577, 579–580
processes 532, 540, 544, 586, 611, 626
steps 540, 570
wizard 500, 507, 524, 527, 532–533, 545–546
integrated
applications 59, 120
building applications 59
custom services 146
services 38, 120, 146
Integrated Services Environment 38
integration
debugger 134–135
service module 432
interface
editor 370, 373
Export 123
icon 418, 421, 423, 426, 452–454
imported 141
map 124
Name 369
names 390
selection 384–385, 388
support 124
interfaces
Internet Protocol 4
Interworking 29–30, 34
INVITE 4, 8, 11–12, 14–15, 18, 21, 24, 83, 86, 109, 176, 208–209, 214–215, 220–221, 231, 234, 245–246, 264, 274–276, 296, 323, 325, 329–330, 332, 336, 349, 354, 357, 474–475, 484–485
Invocation
service method 207
invokeBPEL 358–359, 482, 484
invokeCallBack interface 371–372
invokeCallBackResponse 333
IOException 210, 217, 227, 242–243, 246, 263, 269, 274–276, 278, 355–357, 360
IP multimedia sessions 29
IP Multimedia Subsystem 20, 27–28, 61, 161, 174–175, 178, 180, 440, 443, 541
See also IMS

IP telephony 4
ISC 34, 49, 58–59, 83, 101–105, 162, 164, 174–178, 440
    interface 101, 174–177
    See also IMS Service Control
    SIP servlet sample 101–102
ISCDemo 101, 107–108
ISCDemoApp 105, 107–109
ISCDemoAppHandler 108–110

J

J2EE 49, 54–55, 59, 65–66, 124, 126, 165, 229, 238, 279–280, 294, 316
    See also Java 2 Platform, Enterprise Edition
J2EE applications
    deploy 66
J2EE Connector Architecture 59
J2ME 20
JAIN 20–23
    SIP 20–23
    SIP API 20–23
Java 2 Platform, Enterprise Edition 54–55
    See also J2EE
Java Build path 97, 113–115, 118
Java Community Process 20, 22–23, 215
    See also JCP
Java component 124
Java Message Service 52, 60, 201, 554
Java snippets 135
JavaServer Pages 123
JCP 20, 170, 215
    See also Java Community Process
JDBC 182, 540, 561, 563–565, 569, 571–572, 574–575, 577, 579–580, 586, 594–596, 618–619, 624–625
JMS 52, 60, 126, 167, 201, 554, 557–561, 570, 576, 616–617
JNDI
    name 558–559, 561, 565, 569, 571–572, 574–575, 577, 579, 596, 602–603, 616–617, 619
    name field 558–559, 561, 565, 596, 616–617, 619
JSP 228, 264–268, 571
JSR 23, 55, 64–66, 163, 165–167, 170
Jython development tools 64

K

key concepts 119–120

L

languages 54, 124
latency 169, 230, 311
latitude 321, 337, 339, 404, 435–436, 450
level services 55, 57
    compose foundation 55
Level.INFO 243–246, 263, 270, 274–278
libraries 55, 58–59, 90, 96–98, 112–117, 130
Linux
    Red Hat Enterprise 3.0 88, 143
    server 442
    SuSE Enterprise Server 9 143
    test server 440, 442, 447–449, 452–455, 457–458, 471–472, 478, 481–484, 486, 493, 495–496
    test server logfiles 496
List
    drop-down 558, 563, 565, 576, 595–596, 616, 618–619
    resource 183, 192, 467
Listener 20, 75, 80–82, 223–226, 575
    class 225–226
    example 225
ListeningPoint 21–22
ListIterator 211
localhost
    server 290, 294
localHostIpAddress 465–466
localHostname 354, 465–466
localPortNumber 354
localProcessServer 354
LocalProxy 217, 234–235, 238, 245–246, 255, 342
    servlet 234, 246
    SIP Servlets 235
location
    current 234, 304, 321, 324, 337, 436
location server
    class diagram 436
    configuration 374–375
    set 448
    set up 448
    simulated 441, 448, 483, 496
location service
    WSDL file 373
location simulator 321, 341–342, 399, 435–436

LocationServer 377
EAR 374, 376, 448, 452
LocationServicePartner 388, 399, 404, 425
LockOutServices 313, 323–324
group 313
log
file 481–482, 484, 493–496
log.info 354–360
Logger
getLogger 243–244, 246, 263, 269, 354
log 243–246, 263, 270, 274–278
logic
custom 128
developing application 226
low latency applications 311

M
Ma (interface) 34
makeCall 153, 337, 339, 380, 407–411, 415, 480, 582, 585
activity 410
MakeCallServiceException 410
management
domain 160
enabler 184–185, 188–189, 191
Mandatory mediation primitives 148
Policy/Subscription 148, 195
Service invocation 148, 195
Transaction recorder 148, 195
Manual 134, 140
map security roles 571–572, 574, 576–577, 579–580
Mapping
rules 207–209, 219, 221
maps 9, 14, 32, 128, 134–135, 190, 363
matching reference 425–427
materials
additional 442, 445, 447–448, 450, 465, 503
Max-Forwards 330–331, 334, 470, 475–476
Mb 34, 88
Media Gateway 33, 35–36
See also MG
Media Gateway Controller Function 33
See also MGCF
Media Resource Function 33
See also MRF
Media Server Function Control 36
Mediation
flow component 122
flows 60, 120, 134–135, 142, 147, 150–151, 154, 157, 194, 524, 527, 529
module 122
primitives 60, 135, 146–150, 154, 156–157, 194
custom 147, 150
service applications 147
services 60, 147
services intercept 60
members
available 357, 359, 435–436
new 378, 380–381
message
body 9–10
content 212–213, 216
messaging
applications 312
instant 4, 23, 25–26, 171, 182, 187
interface 58, 100, 633
Method
stubs 242, 261, 272–273, 350
Mg 33–35
MGCF 33–34, 36
See also Media Gateway Controller Function
Mi 34
Mj 34
Mk 34
Mm 34
Mobile Information Device Profile 20
models 20, 46, 188, 207, 338
Module
map 449, 459, 570, 572, 575, 577, 579–580
Name 362
SIP 70–71
MRF 33
See also Media Resource Function
MRFC 33–34
MRFP 33
multipart mime 331, 336, 357
multiplayer gaming 302, 305
Mw 34
Mx 34
MyImsProcessModule 127
MyService 131
BPEL 131
BPEL process 131
SIP 131

N

Name
field 558–561
port 628–630
server 628
network
elements 162, 197
servers 7, 23
Network Statistics Mediation 148
Network statistics mediation primitive 148
New
Condition 394, 396, 403, 407, 412
Server 280–281, 284, 287
SIP Servlet 239, 342, 344
SIP Servlet Wizard 239
node
choice 394, 396–397, 407–408, 411–413
notifications 59, 98, 180, 183–184, 188–189, 197, 199
NOTIFY 12, 24, 183, 187, 210, 296, 317, 324, 330–332, 336, 355–357

O

Open Service Access 37
Open Services Architecture 37
operation
makeCall 585
operators 29–30, 220
Optional mediation primitives 148
CEI event emitter 149
Group Resolution 149
Network statistics 148
Service authorization 149
SLA enforcement 149
OPTIONS 11–12, 24, 75, 129, 141, 209–210, 222, 250, 296, 305, 311, 325, 449, 459, 461, 464, 488–489, 491, 500, 519, 525, 543–544, 570–573, 575, 577–580
OptionsSipServlet
example 210
order
deployed application 221
originator 9, 11, 14, 333, 337, 339, 356–357, 359, 371–372, 390, 397, 403, 406–407, 435–436, 482
OSA application server 34
output
example trace 296
parameter values 134

P

Packaging 55, 63, 66, 141
packets 53, 181, 489–492, 634
palette 129, 147, 150, 417, 420, 423, 425–426
panel
transport chain 631–632
parameters 72, 347–348
numbered list of 590, 613–614
output 358–359, 372, 386, 401, 404, 409, 414
params 331–332, 446–447, 470
Parlay X 2.1
Resources 59
Parlay X Web Service Interface 325
ParseXML 336, 353, 355, 357–358
parsing 167, 210–212, 332, 335–336
participants 4–5, 28
Partner
selected 400, 404, 409, 414
Payment 59, 197
P-CSCF 32, 34
peer 5, 7, 12, 16, 171, 178, 274–276, 476, 633
remote 633
peer.req 274–276
peer-to-peer 5, 7, 16, 178
permissions 56, 186, 548, 610
Personal Data Assistant 31
PestControl 467–469
P-Headers 177
physical server 193
platform
application 165
service 163, 175–176, 545, 550, 585
service delivery 161
PME 184, 189
point-to-point relationships 25, 222
Policy Subscription 157
Policy/Subscription mediation primitive 148, 195
port number
correct 584
portlets 54, 64, 166–167, 169
portNumberDefault 353, 356
Presence
Resources 59, 90
Presence Server
components 100
deploy applications 611
directory 611
presencePublishT 617
presenceServerHostname 347, 353–354, 356

PresenceServerPortNumber 348, 353–354, 356, 457
presentities 182–184
problem determination 439, 480
Process
AddressList 390–391, 394–397
choice node 396–397
component file 425
ID 455, 485
processor 33, 88, 143, 442–443
processServerURI 348, 353–354, 359, 457, 482
private static string 353
production server 55
products 141–142, 301, 305, 312, 440, 522, 541, 546
profile creation 543, 587–588, 612, 626–627
profiles 281, 481–482, 484, 495–496, 518–519, 554
Program 54, 100, 134, 142, 461, 494, 523, 525
Files 494, 523, 525
Programmatic 134, 140, 226
Project Name 103, 111, 113, 237–238, 259, 279, 343, 582
properties
service-specific 56
protocols 5–6, 28–29, 35–36, 100, 146, 196–197
providers 20, 60, 147, 193, 200, 557, 559–560, 563, 595–596, 616–619
provisional responses 14, 22, 170, 216
Proxy
application 256
object 217–219
Server 5, 8–9, 11, 13–14, 17–18, 21, 32, 167, 171–172, 175, 183
Web 8
proxying 25, 217–219, 246
PS 471–472, 482, 611–612, 614–618, 621–623, 625
application 617, 621–622, 625
application server 617, 625
PsetControl
group 468–469
PSTN 6, 33–36, 146
networks 34
Public Switched Telephone Network 6
See also PSTN
Publish-Subscribe-Notify service 183
PXNotifyBus 555–558, 560–561
PXNotifyQueue 559, 561, 576

Q

QoS 29–30, 35–36, 167, 425
Quality of Service 29
See also QoS

R

Rational
Application Developer 141–142, 166, 525
Functional Tester 141
Performance Tester 141
Software Architect 141, 307, 494
Software Development Platform 141
Software Modeler 141
Web Developer 141
rb 471–472, 478
RC 325, 355, 474, 481
Real-time Transport Control Protocol 6
See also RTCP
reason-phrase 13, 15
receipt 9, 13–14, 207–208, 331, 393, 474
Red Hat Enterprise Linux 3.0 88, 143
RedbookGroup.xml 610–611
Redbooks Web site 645
Contact us xviii
Redirect Server 5, 9, 14, 18, 33
Redirection 9, 14, 212, 216
REFER 12, 23, 34, 123, 142, 188, 208, 215, 244, 296, 316, 325, 361, 427, 480, 482, 517, 540
Reference
Partner 387–390, 399–400, 404, 409, 414
points 28, 30, 33–34, 179
References 73, 75, 82, 123, 125, 127, 134, 200, 387, 425–427, 575
page 75, 82
Registrar
application 244
SIP Servlet 239, 242–243
SipServlet 242
Rename 393, 395–399, 402–408, 410, 412–415
req
createResponse 218, 243, 247
getApplicationSession 275
getParameter 263–264
getTo 218, 243, 246–247, 274
setAttribute 264
Request
getAttribute 265
line 11, 13

messages 10, 21, 25, 208
URI 12, 18
requesters 60, 146–147, 199
Resource List Server 183
resource-lists 191–192, 446–447, 610–611
Resources
Diameter 58, 90
Parlay X 2.1 59
Presence 59, 90
resp.getApplicationSession 274
Response
final 15, 230
informational 24
message 10, 13–14, 21, 23–25, 434
redirection 14, 216
Status 14, 610–611
Status Message 610–611
success 14, 216
retrieve
group information 331
Rf
accounting Web 58, 100, 181–182, 465, 625–626, 633
properties 633
services 98
RFC 4, 11–12, 22–23, 25, 35, 170–171, 174, 230–231, 244
RLS services 192, 467
documents 192, 467
usage 192
xlmns 447
rls-services 191–192, 447–448
Roaming 30
routing
intelligent services 5
service 316
RTCP 6, 35
See also Real-time Transport Control Protocol
RTP 6, 33, 35, 297
Rule 120–121, 124, 132, 135, 172, 201, 207–209, 219–221, 226, 311
group 124, 136
sets 134–135, 201
runtime environment 51–52, 142, 231, 361
runtimes 151, 430–431, 523, 525, 529

S

Sample
application
design 312
environment 449, 471, 478, 481–482, 484, 496
scenario 101, 633
test environment 531, 623–624
application scenarios 502
code 214, 224, 342, 355, 357–360
IMS application 151, 157
test environment 439
IMS foundation application 101
SIP application 85, 233
SIP services 82
test environment 457, 461, 493, 541, 547, 589–590, 600, 613–614
sample
application development 507
service 341
Sample application
blocking 84
Sample SIP application 85, 233
SAR file 71–72, 360, 455–457, 459
SCA 121, 123–126, 130, 200, 312, 348, 353, 482
components 123–124, 126
See also Service Component Architecture
SCIM 302–303, 305
See also Service Capability Interaction Manager
S-CSCF 32–33, 101, 176, 197, 622
S-CSCF server 622
SCTP 35, 179
SDO 35, 121, 200
See also Service Data Object
security
page 75–79
roles 75–76, 226, 571–572, 574, 576–577, 579–580
Selector 124, 129, 136, 190–191
sequence
diagram 234–235, 257–258
Servers
billing 58, 100
Service
Architecture 38, 130, 302, 312
Authorization Mediation 149
building blocks 302, 306
capabilities 30, 39, 146
Capability Server 37
choreography 303–304
engine 304

components 121–122, 130
composition 302–303, 306
configuration information 197
consumers 200
creation 28, 30, 39, 43–45, 51–53, 55–57, 160–161
creation domain 160
creation environment 43, 45, 51, 53, 57, 161
Delivery Platform 161
development 44–45, 159, 161
environment 161
endpoint 336
error 583
exception 435
execution 44, 60–61, 159, 161–162, 164, 173
execution environment 44, 60–61, 159, 162, 164
execution platform 162, 173
full cycle development 44–45
service architecture 130, 312
Service authorization mediation primitive 149
Service Capability Interaction Manager 302
See also SCIM
Service Component Architecture 121, 130, 200, 312
See also SCA
service components 130
Service Creation Environment 43, 45, 51, 53, 57, 161
service creation environment 51, 57
Service Data Object 121, 200
See also SDO
Service delivery environment 161–162
service enablers 39, 57, 61, 162, 164–165, 182, 313
common 182
modular 61, 164–165
prebuilt IMS-compliant 162
service execution environment 61, 164
service FindHelp 469
service identification 306
service implementations 57, 146, 149, 196–199
service interactions 200
service interface 307, 321, 325, 378, 435, 486
service invocation
dynamic 148
Service Invocation Mediation 148
service invocation mediation 148, 195
Service invocation mediation primitive 148, 195
service logic 49, 160
service method 23, 25, 207–208
service modules 132–133
Service Orchestration 170, 302, 305–306, 311
Service Oriented Architecture 165
See also SOA
Service Pack 442
Service plane 312
service plane 36, 39, 160, 174, 176, 179, 312
service plane architecture 312
service plane solutions 312
service platform 163, 175–176, 545, 550, 585
Service Point 176
service policies 146
Service Policy Manager 146, 193, 199, 549
Service Policy Manager Database 549
service provider 149, 194, 197, 313–314, 323
Service Proxy 175
service request 175, 333
service requesters 147
service requirements 44
Service SIP Servlet 132
Service SIP Servlet wrapper 132
service ThirdPartyCall 585
service topic 440, 447, 466–467
service usage 199
service usage record 199
ServiceError 379–380
ServiceException 379–380, 410, 583
service-oriented architecture 57, 162, 165, 170, 193, 200
See also SOA
Service-oriented modeling 306
ServicePlatform 545, 548, 550–551
services
accounting 35, 180
component 120
componentized 120
compose 302
diameter 178, 180
lockout 323
multimedia 28–29, 31
non-IMS 160, 163
services architecture 38–40
services integration 28, 312
services plane 165
servlet
container 23, 25, 168, 206–208, 226
deployment descriptor 68, 240–241, 260, 272,

277, 346–347, 349–350
dialog 260–261, 271
servlet-class 215, 268
ServletConfig 25, 269, 352, 354
ServletContext 25, 72, 213, 225, 227–228, 277, 352–356
ServletException 210, 214, 217, 227, 242–243, 246, 263, 269, 275–277, 354–355, 357, 360
servlet-name 214–215, 221, 228, 268
ServletResponse 269
ServletTimer 225
Session Initiation Protocol 3–4, 22–23, 25, 35, 66, 69, 82, 167, 182
Session management 4–5, 28, 32, 167, 208
Session setup 5
setAttribute 222, 264, 274–278, 355, 357
Sh 34, 59, 97–98, 101, 110, 118, 179–182, 446–447, 541, 547–554, 581–582, 587, 589–590, 600–601, 610–613, 617, 625–626, 628–629
Sh services 59, 98
Sh subscriber profile 59, 98, 181–182
Si 34, 540, 554–557, 570
SI Bus 540, 554–557, 570
See also Service Integration Bus
See also Service Itegration Bus
Signaling gateways 33
SIMPLE 4–5, 17, 19, 26, 53, 55, 60, 83, 86, 123, 139, 166, 171–172, 182, 189, 210, 217, 233–234, 242, 256, 307
Simple Mail Transfer Protocol 5
Simulated CCF 441
simulated FindHelp Caller location 450
Simulated Location 441, 444, 448–449, 483, 496
Simulated Location Server 441, 448, 483, 496
SIP
based services 196
call flow 17, 19
communication 7, 307, 309
components 15, 55
container 23, 69, 167, 169–170, 172, 182, 206–207, 210, 215–217, 221, 229, 294, 297
content 69–70, 72
converged 49, 65–66, 69, 71, 163, 169, 233, 258, 280, 307
developing 43, 50–51, 66, 205, 228, 230, 234
dialogs 16, 171, 208, 273, 309, 353
extension 23, 25, 171, 182
interface 20, 187, 257
phone 85, 87, 209, 318, 329–330, 333, 464, 477
project 55, 59, 67, 71–72, 75, 90–93, 105, 236–238, 258, 335, 342–344, 361
sessions 11, 17, 310
signaling 4, 9, 32, 36, 165
softphone 441, 445, 449, 469–471, 474–475, 477, 485, 505, 507
stack 20–21, 61, 167
tool 85, 87
transactions 15–16
URI 5–6, 176, 192, 254, 330, 336, 354, 357, 359, 445, 473, 492
SIP ACK 24, 325, 474
SIP addresses 337, 583
SIP Application 3, 23, 26, 33–34, 49, 54–56, 63, 65–67, 71–72, 85, 167–168, 170, 173, 205–207, 209, 215, 217, 223, 229, 231, 233–235, 249–250, 256, 304, 440, 478
developing 205, 234
elements of 209
packaging 66
project 235
server 34, 55–56, 173, 304, 440, 478
session 168
support 167, 173
SIP Application Archive 250
SIP Application development 54, 65
SIP Application Resource 66
SIP architectural components 7
SIP AST 500, 502
SIP B2BUA 33, 37
SIP deployment descriptor 72
SIP deployment descriptor editor 66, 72–73
SIP Facets 92
SIP INVITE message 8, 12, 18, 325
SIP messages 9–10, 32, 36, 53, 66, 168, 172, 174, 206–207, 230, 294, 318–320, 322, 471–472, 477–478, 488
SIP method 12, 24, 100, 171, 176, 182–184, 187, 189, 197, 210, 296, 324, 330, 354–357, 622
BYE 11–12, 19, 24, 208–209, 272–276, 278, 296, 325, 334, 360, 474–475, 477
CANCEL 8, 11–12, 24, 208–209, 215, 217, 264, 296
INFO 12, 22, 24, 101, 126, 170–171, 177, 210, 225, 243–246, 263, 270, 274–278, 294, 296, 324–325, 334, 354–360, 474, 476, 484, 487, 491
NOTIFY 12, 24, 183, 187, 210, 296, 317, 324,

330–332, 336, 355–357
OPTIONS 11–12, 24, 75, 129, 141, 209–210, 222, 250, 296, 305, 311, 325, 449, 459, 461, 464, 488–489, 491, 500, 519, 525, 543–544, 570–573, 575, 577–580
REFER 12, 23, 34, 123, 142, 188, 208, 215, 244, 296, 316, 325, 361, 427, 480, 482, 517, 540
REGISTER 9, 11–12, 17–18, 21, 24, 176, 180, 183, 209, 211, 216, 222–223, 234, 240–243, 266, 286, 316, 330
UPDATE 12, 15, 23, 59, 143, 180, 182, 242, 244, 262–266, 268, 442, 463, 489, 509, 521–523, 586, 611, 622, 626
SIP Module 70–71
SIP network server 7
SIP Parlay based services 196
SIP protocol 13, 20–23, 25, 53, 124, 207–208, 213, 215, 229–230
SIP proxy 8–9, 11, 21, 32–33, 37, 171–172, 175, 197, 212, 231, 586
server 8–9, 11, 21, 32
SIP Redirect Server mode 33
SIP Register message 9
SIP requests 11–12, 25, 175, 177, 207, 209–210, 216, 219, 222, 272
SIP response messages 13
SIP responses 13, 25, 208, 216
SIP servers 5, 32, 36, 59, 462
SIP Servlet 23–26, 49, 55, 59, 65–69, 71, 74, 76–77, 83, 86, 101–102, 107, 124–126, 131–132, 163, 166–167, 169–170, 205–209, 212–213, 216–219, 221–226, 228–231, 235, 238–243, 245, 247, 270–271, 273, 277, 307–311, 313, 328–329, 332–338, 341–346, 348, 350–354, 361, 368, 420, 427, 430, 440, 442, 444, 455, 458–459, 478, 481–484, 496
API 23–24, 59, 67, 101, 205–207, 209, 222
API specification 206
applications 49, 65, 69, 228–230
client 420
code 311, 352
container 206–207, 226
create 239–241, 271
deploy 166
deployment 68, 71, 277
deployment descriptor 68, 277
deployment descriptor information 68
design 328
developing 205, 228, 230
developing applications 228
development 65, 342, 361
FindHelp 478
implementation 335
interact 313
listener types 224
methods 206
pass-through 308
proxies 218
SIP Servlet method
doAck 24, 107, 209
doBye 24, 107, 209, 273, 275
doCancel 24, 209
doError 107
doErrorResponse 24, 216, 273
doInfo 24, 107, 210
doInvite 24, 107–108, 208–209, 246, 350, 354–355, 481
doMessage 24, 210
doNotify 24, 210, 350, 357–358, 482
doOptions 24, 209–210
doPrack 24, 210
doProvisionalResponse 24, 216, 273
doRedirectResponse 24, 216
doRegister 24, 209, 242–243
doResponse 23, 25, 107, 216
doSubscribe 24, 210
doSuccess 107
doSuccessResponse 24, 208, 216, 273–274, 289, 350, 360
SIP User Agent 7–8, 16, 21, 252
client 307
server 37
SIP Web Services 305, 307, 310
SIP Working Group 4
SipAddress 25
SipApplicationSession 25, 213–214, 222–225, 227–228, 231, 263, 268, 274, 276, 278
appSession 214, 225, 228, 263, 274, 276, 278
expiration 223
links 222
object 227
session 224
SipApplicationSessionEvent 224
SipApplicationSessionListener 223–224
example 224
interface 223
sipFactory 25, 213–214, 227, 262–263, 275–277, 353, 355–356

createURI 214, 263
SipListener 21
interface 21
SIPMessageFactory 296
SIPmethod
ACK 11–15, 18–19, 24, 209, 217, 224, 275, 296, 325, 474, 476
INVITE 4, 8, 11–12, 14–15, 18, 21, 24, 83, 86, 109, 176, 208–209, 214–215, 220–221, 231, 234, 245–246, 264, 274–276, 296, 323, 325, 329–330, 332, 336, 349, 354, 357, 474–475, 484–485
SIPModule 92
SIPp 53, 85, 87, 316, 318, 329–330, 334, 354, 437, 441–442, 445, 460–461, 469–474, 477–479, 481–482, 485, 494
installation 460–461
logs 494
script 316, 329, 474, 477–478, 481, 485
SIPphone 329–330, 333, 437
SIPPING 4, 171
sipPort 356
SipProvider 21–22
SIPReadHandler 296
SipServlet 25, 67, 209–210, 213–215, 217, 227, 242–243, 246, 263, 270–272, 277, 294, 353
SipServletMessage 25, 210, 212, 227, 336
SipServletMessage interface 210, 212, 227
SipServletRequest 23, 25, 207–208, 210, 214–215, 217, 243, 246, 274–276, 355–357, 360
createRequest 214–215
request 207–208, 210, 217
SipServletResponse 23, 25, 207–208, 210, 212–213, 217, 243, 246–247, 274–275, 355, 358, 360
classes 23
peerResp 275
response 207–208, 210
ringing 355
SC 212, 243, 247
SipSession 25, 208, 213–215, 222–224, 231, 273, 275–276, 278, 353, 360
dialog 275
interface 208, 222
SipStack 21
SIPUdpConnect 296
SIPUdpConnection 296
SipURI 219, 263, 274, 276, 353, 356
SIP-version 13
sipXphone 53, 252, 254–256, 286–287, 289–290, 292, 318, 441, 460–464, 474, 494–495, 503–505
installation 252, 504
SJphone 441, 460, 464, 495, 507–508
SLA enforcement mediation primitive 149
SLF 32, 34, 179
See also Subscriber Location Functions
SMS Service 131
SOA 57, 162, 165, 170, 193, 200
See also Service Oriented Architecture
See also service-oriented architecture
SOAAdministrator 571–572, 574, 576–577, 579–581
SOAO 57, 120, 162, 165, 199–201, 306, 312, 583–586
SOAP 60, 167, 333, 421, 432, 583
SSL 629–630, 632–633
Stand-alone references 123
standardized interfaces 20, 30, 60
State machine 120, 124, 134–136, 201, 229, 319
Stateless SIP Proxy 212, 231
Status
code 13–15
line 13
OK 407–408, 410, 412–415
SUBSCRIBE 12, 24, 100, 171, 176, 182–184, 187, 189, 197, 210, 296, 324, 330, 354–357, 622
subscribe 12, 24, 100, 171, 176, 182–184, 187, 189, 197, 210, 296, 324, 330, 354–357, 622
message 330, 354
requests 24, 182–183, 622
Subscriber Location Functions 32
See also SLF
subscriber profile information 181
Subscriber Profile Services 180
subscribers 30, 183, 187, 315–316, 319–320, 322
subscription 26, 32, 34, 59, 100, 148, 171, 179, 183–184, 186, 195, 324, 330–331, 481, 617
Substitute 447, 471–472, 478
Success 14, 213, 216–217, 274, 323–324, 537
SuSE Linux Enterprise Server 9 143
SystemOut.log 495, 583, 585
Systems Management Services 160

T

TCP 168, 179, 212, 253, 331, 462, 471–472, 477–478, 628, 630–631
Telecom Web Services

Access Gateway 146, 148–149, 193–195
Server 61, 145–146, 150–151, 165, 193–194, 196–197, 316, 322, 339, 379, 440, 443, 524, 540
Toolkit 50–51, 57, 60, 119
Telecommunications Technology Association of South Korea 28
Terminal Status 59, 198
TerminalLocationImpl 339, 377–378, 389, 423
TerminalLocationImport 423–425, 452–453
component 423
Properties 452
Test
component 137, 433
editor 138–140, 433
environment 59, 134, 166, 252, 294, 430, 439–442, 444–445, 455, 457–458, 461, 466, 493, 517–519, 531, 541, 547, 589–591, 600, 613–614, 623–624
SIP applications 66, 85
TestServer
jar 465
Third Party Call
application 177, 222, 256–258, 286–287, 289
Control 259–260, 265–266, 277, 279, 287, 289, 337, 425, 453
Control Partner 390, 409, 426
Controller 214–215, 277
Import 425–426, 453
component 426
Properties 453
interface 153, 339, 410
Partner 426
Server 87, 263
Web Service 580, 586
TimerListener
interface 225
TimerService
object 225
TISPAN 29
toSipURI 263–265, 277
Transaction recorder 155–157
mediation primitive 148, 195
transactions 15–16, 20, 178, 180
Transport 6, 28, 33–36, 160, 179, 212, 216, 226, 312, 331, 421–422, 470, 475–476, 630–632
plane 36, 160
TWSS
Administration Console 583–586
application 583
Components 545
Core Web Services 584
default message flow 149–150
deploy applications 540, 570

U

UDP
message 481
port number 471–472
UE 31–32, 34
See also User Equipment
UMTS 29
See also Universal Mobile Telecommunications System
Unified Service Creation Environment 45, 161
See also USCE
Uniform Resource Identifier 5
See also URI
Universal Mobile Telecommunications System 29
See also UMTS
UPDATE 12, 15, 23, 59, 143, 180, 182, 242, 244, 262–266, 268, 442, 463, 489, 509, 521–523, 586, 611, 622, 626
URI 5–6, 12, 18, 171, 176, 185, 190–192, 214, 217, 219–221, 231, 244–247, 254–255, 324, 329–332, 336, 339, 348, 353–355, 357, 359, 400, 404, 435–436, 445–447, 450, 468, 473, 482, 492, 583–584, 623, 631–632
See also Uniform Resource Identifier
USCE 45–50
See also Unified Service Creation Environment
Use data Type Variables 400, 404, 409, 414
User
capabilities 5
Databases 32
group 593–594, 607
ID 282, 547–548, 562, 589, 592–593, 600–601, 613, 628
information 36, 464–465
location 5
service 302, 306
User Agent
Back-to-Back 8, 37, 59, 175
Client 8, 14, 213, 215–216, 307–308, 310
Server 8, 14, 37, 183, 302, 305
User Equipment 31, 36
See also UE
Ut 34, 187

UTF-8 215, 268, 330–334, 358, 446–447, 470, 475, 481, 610

V

video 4–6, 29, 35, 162
Visual snippet 129–130, 134–135, 393, 395–396, 398, 402–403, 405–408, 410–411, 413–415
voicemail server 230

W

WAS
  profile 586–587, 611, 626
  See also WebSphere Application Server
Web
  application 70, 72, 166, 268, 318
  deployment descriptor 69, 72–73, 75–76, 228, 267–268, 277
  interface 85, 87, 199, 256–257, 461
  proxy 8
  server 5, 505
  sites 335
  tools 54, 58, 64, 78
Web Service
  application 100
  call flow 329
  code 335
  consumer 310, 435
  Interface 307, 321, 325, 435, 486
  operations 149
  Ports 378, 380–381
  Properties 582
  requests 146, 181
  response 333
Web Services
  administrative 199
  connectivity 52
  core TWSS 583
  endpoint 126
  Explorer 582
  Implementation 197
  Interface 180
  Platform 584
Web Services Description Language 60, 124, 182
  See also WSDL
web.xml 69, 71–72
WEB-INF 69, 72, 377, 379, 381
WebSphere
  Application Server Toolkit 50–51, 53, 55, 63–64, 89, 233–234, 236, 319, 342, 455, 500–501
    install 236, 500
  Diameter Enabler 180–181, 340, 380–381, 625, 633
  Enterprise Service Bus 52, 60–61, 120, 122, 142, 148, 165, 194, 199–201, 312, 518
  IMS Connector 56, 164, 173
  Integrated Solutions Console 592, 594–596, 602, 604, 606–607, 615–619, 621–622
  Message Broker 312
  Telecom Web Services Server 61, 150–151, 165, 194, 316, 322, 339, 379, 440, 443, 540
WebSphere Application Server
  install 532
  integrated 525
  LaunchPad 500
  Network Deployment 172, 185, 200, 500, 532–533, 625
  Node 448, 458
  product 167
  profile 586–587, 611, 626
  Service Integration 554
WebSphere Integration Developer 57, 59–60, 119–121, 123, 126, 132, 141–143, 145, 150, 194, 200–201, 311, 320, 341–342, 361, 376, 430, 450, 477, 494, 515–517, 521–524, 526
  See also WID
WebSphere Process Server
  Console 479, 482, 486
  EAR 621
  key abstractions 121
  support 121
WID 119–120, 124, 130, 134, 136, 146, 150–151, 494, 521, 526–527, 529
  See also WebSphere Integration Developer
Windows
  2000 88, 142
  2003 142
  XP 88, 143, 442
  XP Professional 143, 442
Wiring
  Advanced 425–427
WLAN 29, 31
WPS
  console 483, 486
WPSSetDB2Tables.sh 612–613
WS-Addressing service 383
wsadmin 166, 571–574, 576, 578–579, 581, 600,

628
WS-BPEL 122, 201
WSDL
files 65, 90, 123, 128, 133, 364, 373, 375, 379–381, 387, 427, 430, 582
See also Web Services Description Language
WTP 58, 509–511, 514–515

X

XCAP
Application User IDentifier 191
interface 187, 190
See also XML Configuration Access Protocol
server 190, 192–193
User Identifier 191–193
XDM 190, 192, 609
See also XML Document Management
X-Lite 252–255, 286–287, 289–290, 292–293, 297, 318, 505–506
installation of 505–506
XML
body 330, 336, 357
documents 180, 187, 190
XML Configuration Access Protocol 49, 190
See also XCAP
XML Document Management 190
See also XDM
xmlHttp 265–266
XUI 191–193

IBM

Redbooks

Developing SIP and IP Multimedia Subsystem (IMS) Applications

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

Developing SIP and IP Multimedia Subsystem (IMS) Applications

Redbooks

Hands-on introduction to toolkits and development environments

Learn to develop converged and composite services

Programming guidelines and working examples

The convergence of Internet Protocol (IP) networks is enabling seamless communications that combine data, voice, video and other information streams. The true value of converged IP network however is realized through the converged applications that leverage the network. The key enabler to developing converged applications is the platform for designing, developing, testing, and deploying applications that integrate and compose services.

This IBM Redbook introduces IBM tools for creating converged Session Initiation Protocol (SIP) and IP Multimedia Subsystem (IMS) applications. It provides programming guidelines and working examples that demonstrate how to use the different development tools. It also provides hints and tips that enable you to quickly get up to speed developing converged applications.

The portfolio of products include the IBM WebSphere Application Server Network Deployment, IBM WebSphere IP Multimedia Subsystem Connector, IBM WebSphere Presence Server, IBM WebSphere Telecom Web Services Server, and IBM WebSphere Integration Developer.

This redbook is aimed at the diverse set of professionals that design and develop SIP and IMS applications.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7255-00 ISBN 0738489573