Building SOA-based Solutions for IBM System i Platform

Implementing service-oriented architecture (SOA) with Web services

Examples of Web services based on ProgramCall, HATS, PHP, and Web Services Client for ILE

Excellent starting point for System i developers on Web services

Daniel Hiebert
Rolf André Klaedtke
Elena Lowery
Aleksandr Nartovich
Nitin Raut
Michael J. Sandberg

ibm.com/redbooks
Note: Before using this information and the product it supports, read the information in “Notices” on page vii.
Contents

Notices ........................................................................................................ vii
Trademarks ................................................................................................. viii

Preface ......................................................................................................... ix
The team that wrote this book ................................................................. ix
Become a published author ................................................................... x
Comments welcome .............................................................................. xi

Part 1. SOA: Understanding the big picture ........................................... 1

Chapter 1. SOA overview ...................................................................... 3
  1.1 A simple definition of SOA ............................................................... 4
    1.1.1 Defining SOA ........................................................................... 4
  1.2 Introducing services ......................................................................... 4
    1.2.1 A coffee making machine based upon services ....................... 5
    1.2.2 A non-technical but business-related example ....................... 5
  1.3 SOA characteristics ......................................................................... 6
    1.3.1 Implement SOA in many different ways .................................. 6
    1.3.2 Take advantage of loose coupling .......................................... 7
    1.3.3 Continue to use the existing applications ............................... 7
    1.3.4 Implement quality of services ................................................. 8
  1.4 SOA from a business perspective .................................................... 8
    1.4.1 Reasons to consider SOA ......................................................... 8
    1.4.2 SOA is not always a perfect fit .............................................. 9
  1.5 Further reading ................................................................................ 10
    1.5.1 IBM Redbooks publications ................................................... 10
    1.5.2 Web sites ............................................................................... 10

Chapter 2. SOA application design ....................................................... 11
  2.1 Designing an SOA solution ............................................................... 12
    2.1.1 Designing services ................................................................ 12

Chapter 3. Web services technology stack ........................................ 17
  3.1 Web services technologies in action .............................................. 18
  3.2 SOAP: Web services messaging layer ............................................ 19
  3.3 Web services transport ................................................................. 21
  3.4 Web services description: WSDL .................................................. 22
  3.5 Web services discovery: UDDI ...................................................... 24
  3.6 Basic Profile .................................................................................. 24
  3.7 Summary ....................................................................................... 25

Chapter 4. Sample scenario ................................................................. 27
  4.1 Introducing the existing environment ............................................ 28
    4.1.1 Overview of the Flight400 application .................................. 28
  4.2 Facing new challenges ................................................................. 28
    4.2.1 New opportunities ................................................................ 28
    4.2.2 Technical issues to address ................................................... 29
    4.2.3 Flight400: From monolithic System i application to SOA ....... 30
  4.3 Conclusion ..................................................................................... 32
Part 3. Implementing Service Consumer .................................................. 193

Chapter 7. HATS Web service ............................................................... 115
7.1 Project investments in developing a service ..................................... 116
7.1.1 Analyzing an existing application ............................................. 116
7.1.2 Naming conventions ............................................................... 126
7.2 Developing a HATS Web service .................................................... 126
7.2.1 Creating a HATS project .......................................................... 126
7.3 Recording macros ........................................................................... 130
7.4 Setting connection properties ......................................................... 155
7.4.1 Enabling pooling ...................................................................... 155
7.5 Creating the integration object ....................................................... 158
7.6 Creating Web service support files ................................................ 160
7.7 Creating Web service ..................................................................... 163
7.8 Testing the Web service ................................................................. 165
7.9 Next step ......................................................................................... 169
7.10 Summary ....................................................................................... 170

Chapter 8. PHP Web service ................................................................. 171
8.1 Introducing PHP .............................................................................. 172
8.2 Technology overview ...................................................................... 172
8.2.1 How PHP works ....................................................................... 172
8.2.2 What is needed to use PHP ....................................................... 173
8.2.3 There is more than one way to say “Hello, World” ..................... 173
8.3 PHP on the System i platform ........................................................ 177
8.3.1 Zend Core for i5/OS ................................................................. 178
8.3.2 PHP version and availability ................................................... 178
8.3.3 Accessing DB2 UDB and i5/OS resources ................................ 178
8.3.4 Support for Zend products on i5/OS ......................................... 179
8.4 The PHP Extension and Application Repository ............................ 179
8.4.1 Why PEAR important for you .................................................. 179
8.4.2 Installing PEAR packages ....................................................... 180
8.4.3 Further information on PEAR .................................................. 180
8.5 Creating a Web service with PHP ................................................... 180
8.5.1 PHP SOAP implementations .................................................... 180
8.5.2 Zend Studio for i5/OS WSDL Generator ................................ 180
8.5.3 SOAP cache ............................................................................ 181
8.5.4 Creating a simple Web service from PHP ............................... 182
8.5.5 Creating a PHP Web service to wrapper a program call............. 187

Chapter 9. IBM Web Services Client for ILE (RPG, C, C++, COBOL) ... 195
9.1 RPG as a Web service Client .......................................................... 196
9.1.1 Development environment ...................................................... 196
9.2 ProgramCall bean example ............................................................ 197
9.2.1 Opening the J2EE Perspective ................................................... 198
9.2.2 Examining the WSDL document ............................................. 198
9.2.3 Moving the WSDL file to the server ....................................... 200
9.2.4 Using Web Services Client for ILE to generate WSDL artifacts 204
9.2.5 Creating an RPG program to invoke a Web service .................. 204
9.2.6 Compiling RPG ...................................................................... 216
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2.7 Compiling C modules and Create service program</td>
<td>217</td>
</tr>
<tr>
<td>9.2.8 Invoking the RPG application to make reservation</td>
<td>218</td>
</tr>
<tr>
<td>Chapter 10. JSF Web service client</td>
<td>219</td>
</tr>
<tr>
<td>10.1 Developing a JSF client</td>
<td>220</td>
</tr>
<tr>
<td>10.1.1 Creating a dynamic Web project</td>
<td>220</td>
</tr>
<tr>
<td>10.1.2 Creating a page template</td>
<td>220</td>
</tr>
<tr>
<td>10.1.3 Creating JSF Web service client using Web Service Component</td>
<td>231</td>
</tr>
<tr>
<td>10.1.4 Testing the JSF Web service client</td>
<td>239</td>
</tr>
<tr>
<td>10.2 Summary</td>
<td>240</td>
</tr>
<tr>
<td>Chapter 11. PHP Web service client</td>
<td>241</td>
</tr>
<tr>
<td>11.1 Consuming a Web service with PHP</td>
<td>242</td>
</tr>
<tr>
<td>11.1.1 Using Zend Studio for i5/OS to create Web services clients</td>
<td>242</td>
</tr>
<tr>
<td>11.1.2 Consuming the Repeater PHP Web service</td>
<td>243</td>
</tr>
<tr>
<td>11.2 Consuming the ProgramCall PHP Web service</td>
<td>245</td>
</tr>
<tr>
<td>11.2.1 Consuming the GETFLIGHTINFOServices WebSphere Web service</td>
<td>246</td>
</tr>
<tr>
<td>Appendix A. Setting the connection to WebSphere Application Server V6.0</td>
<td>249</td>
</tr>
<tr>
<td>Changing a JMX connection with a server</td>
<td>250</td>
</tr>
<tr>
<td>Appendix B. URI length limit of 259 characters on Windows</td>
<td>255</td>
</tr>
<tr>
<td>Issue</td>
<td>256</td>
</tr>
<tr>
<td>Cause</td>
<td>256</td>
</tr>
<tr>
<td>Solution</td>
<td>256</td>
</tr>
<tr>
<td>Appendix C. Useful tools</td>
<td>267</td>
</tr>
<tr>
<td>SOAP monitoring utility</td>
<td>268</td>
</tr>
<tr>
<td>Download Apache TCPMON binary</td>
<td>268</td>
</tr>
<tr>
<td>TCPMON uses in debugging services problems</td>
<td>273</td>
</tr>
<tr>
<td>TCPMON uses for security</td>
<td>273</td>
</tr>
<tr>
<td>WebSphere Development Studio client - debugger</td>
<td>274</td>
</tr>
<tr>
<td>Using the WebSphere Development Studio client debugger</td>
<td>274</td>
</tr>
<tr>
<td>Appendix D. Additional material</td>
<td>281</td>
</tr>
<tr>
<td>Locating the Web material</td>
<td>281</td>
</tr>
<tr>
<td>Using the Web material</td>
<td>281</td>
</tr>
<tr>
<td>How to use the Web material</td>
<td>281</td>
</tr>
<tr>
<td>Related publications</td>
<td>283</td>
</tr>
<tr>
<td>IBM Redbooks publications</td>
<td>283</td>
</tr>
<tr>
<td>Other publications</td>
<td>283</td>
</tr>
<tr>
<td>Online resources</td>
<td>283</td>
</tr>
<tr>
<td>How to get IBM Redbooks publications</td>
<td>284</td>
</tr>
<tr>
<td>Help from IBM</td>
<td>284</td>
</tr>
<tr>
<td>Index</td>
<td>285</td>
</tr>
</tbody>
</table>
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:

- Redbooks (logo)
- eServer™
- iSeries®
- i5/OS®
- AIX®
- AS/400®
- DB2 Universal Database™
- DB2®
- IBM®
- OS/390®
- OS/400®
- Rational®
- REXX™
- RPG/400®
- System i™
- System i5™
- WebSphere®
- Workplace™

The following terms are trademarks of other companies:

- SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.
- EJB, Java, JavaServer, JDBC, JDK, JMX, JSP, JVM, J2EE, Solaris, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- Internet Explorer, Microsoft, Visual Basic, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- 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

There is a strong shift in the industry toward reuse of the existing software and hardware resources within the companies to minimize the IT cost. Instead of creating or buying a new solutions, companies are trying to build a set of reusable software components based on the existing applications. These components can be quickly assembled in many different ways to satisfy the business needs of the companies. This environment is based on service-oriented architecture (SOA) and solutions that support business process automation.

This book provides the detailed information about multiple ways for building SOA-based solutions around the System i™ platform. The discussion in the book covers the server and client side implementations that include:

- ProgramCall in IBM® Toolbox for Java™
- Host Access Transformation Services (HATS)
- DB2® Web services
- PHP
- IBM Web Services Client for ILE
- Java-Server Faces (JSF)

Parts of the book are appropriate for CIOs, system architects, and application developers.

The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Rochester Center.

**Daniel Hiebert** is an Advisory Software Engineer with IBM Systems and Technology Group. He works with the Web services and the SOA runtime within WebSphere® Application Server on System i. He has also authored several articles and technical papers on application integration through SOA and Web services. He has three years of experience with Application Integration connecting RPG with Java, Web Clients, C#.NET, and DB2 technologies and six years of experience with J2EE™ technologies including JSP™, JSF, Servlets and EJB™.

**Rolf André Klaedtke** is an IBM Certified Application Developer and owner of RAK Software, Consulting and Publishing in Kreuzlingen, Switzerland. He has a commercial background and accumulated over 20 years of experience in the IT industry, initially starting as a software developer using RPG and CL on IBM S/38. Since then, he has used a wide array of DBMS, tools, and languages. In the recent years he concentrated on Web development, mostly using PHP and CSS, as well as on Java and C#.NET. He has co-authored IBM Redbooks® publications, has written for magazines such as PowerTimes, and has organized technical conferences and user group meetings in Switzerland. He can be contacted through his company's Web site at [http://www.raksoft.ch](http://www.raksoft.ch).

**Elena Lowery** is a Technical Consultant in the IBM eServer™ Solutions Enablement organization at IBM Rochester. She helps IBM Business Partners implement various WebSphere technologies on the System i platform. Her areas of expertise include WebSphere Application Server, WebSphere Portal Server, Web services, and Java development.
Aleksandr Nartovich is a Senior IT Specialist in the IBM ITSO, Rochester Center. He joined the ITSO in January 2001 after working as a developer in the IBM WebSphere Business Components organization. During the first part of his career, Aleksandr was a developer in AS/400® communications. Later, he shifted his focus to business components development on WebSphere. Aleksandr holds a degree in Computer Science from the University of Missouri-Kansas City and a degree in Electrical Engineering from Minsk Radio Engineering Institute.

Nitin Raut is an Advisory Software Engineer at System i5™ Technology Center, e-Business Team, located in IBM Rochester, MN. He has been involved in e-Business consulting and education targeting the System i platform for past seven years. His expertise include WebSphere Application Server, WebSphere Portal Server, WebSphere/Rational® Development Studio, WebSphere Business Integration, WebSphere Host Access Transformation Service, WebFacing, Apache, and so forth. He has 18 years of experience which includes assignments in SAP® Basis Consulting, Enterprise Application Development and ERP (BPCS & Mapics XA) Implementation on the System i platform.

Michael Sandberg is a technical consultant working for IBM ISV Business Strategy and Enablement, located in Rochester, Minnesota. For the past five years, he has been involved in supporting System i solution providers as they enhance and innovate their applications. As part of this work, he has accumulated technical expertise in the IBM WebFacing Deployment Tool with HATS Technology, PHP: Hypertext Preprocessor (PHP), and other technologies that focus on application innovation for the System i platform.

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

Nadir Amra
Tony Cairns
Pat Fleming
Kent Milligan
IBM Rochester, MN

Holt Adams
IBM Boulder, CO

Grant Hutchison
IBM Toronto, Cananda

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will 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 books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online Contact us review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an e-mail to:
  redbooks@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
SOA: Understanding the big picture

In this part, we introduce service-oriented architecture (SOA). It includes the overview of the architecture and its benefits, application design methodology, and Web services (which is a major set of standards for implementing SOA).
Chapter 1. SOA overview

This chapter provides an introduction to service-oriented architecture (SOA) and discusses Web services as one of the specific technologies that are associated with it. SOA is a wide field, and it is impossible to cover all related themes herein. Therefore, we also include a list of references that allows you to find more material on this vast topic.
1.1 A simple definition of SOA

With architecture, from the latin word *architecture*, we identify the art and science of designing buildings and structures. This type of design generally includes the creating of complex systems, which probably nobody would start without having a good idea about the underlying structure or architecture. SOA is a concept that allows various approaches to realize a system designed according to its guidelines. In this book we concentrate on a technology that is popular today, Web services.

1.1.1 Defining SOA

SOA is a methodology that you can use in software development. Just as the concept of an *object* is central to object-oriented architecture, SOA is based on the concept of a *service*. In SOA, a service is an application that can be invoked by other applications.

The concept of a service in software is similar to a concept of a service in real life. For example, if you want to relocate, you look up available relocation services in the telephone directory. Then you can contact the relocation company directly and work out the details of the move. The same scenario can apply to a software interaction. A company can have software that handles relocation procedures. The software can search a public service registry for relocation services. Based on some selection criteria (for example price or delivery time), the relocation application can choose one of the relocation providers and work with them directly to schedule the move and arrange payment.

The described services scenario involves the following three participants:

- **Service Requestor**: An application with a business need
- **Service Broker**: A registry of all available services
- **Service Provider**: An application that implements a business function

A *service provider* implements a service and publishes it to the *service broker*. A *service requestor* searches the registry to find a service of interest. Upon finding a service, the service requestor binds to the service provider and invokes the service with the help of an XML file that describes the Web services interface.

1.2 Introducing services

In May 2006, a search for the keyword *Web services* on the Internet brought back over 4 200 000 000 documents. We do not know on how many of them you can find an explanation of Web services. Sometimes the concepts are explained in a language that is difficult to understand. Explaining technical concepts to an unknown audience is indeed a challenging endeavor, especially if the background of that audience varies from complete beginner to expert.

Therefore, to introduce the concept of services and to relate that concept to your applications, we provide an example based on a coffee-making machine. We then follow this simple concept with a non-technical example to introduce you to the topic of Web services.
1.2.1 A coffee making machine based upon services

Here, we use a coffee making machine example for some initial considerations on services in general.

Imagine you are standing in front of a coffee making machine, ready to drop some coins into it, select the appropriate button, and after a little while served a coffee. See this serving of coffee as a service that the machine is providing to you. But, what type of coffee? With sugar and milk or just black? Big or small? There are many possible variations, so let us have a look behind the scenes.

After you drop the coin, you can select a few choices by selecting some buttons to have the machine create exactly the coffee that you want. Each selection triggers some action inside the machine:

- Heat water
- Prepare coffee powder
- Add milk
- Add sugar

Each action needs to be a single service because people might want to have milk and sugar or just one or the other. (As an analogy in IT, you can influence the results that a particular application provides by using different services.)

Now, imagine this coffee making machine is an older model, and you do not have many choices. Either you get a big coffee with milk and sugar or you do not get anything. If this were the case, customers would not be very happy. So, you might have to either replace or update the machine. The same is true with some older applications that are important to your business but that might not satisfy new needs or changing business requirements.

But just how much modernization, respectively cutting into single services do you need? Well, you might decide that heating water and pressing it through the ground coffee could be a single step. But what if later you would like to offer hot tea as an option from the same machine?

The same is true with your application: good judgement and a good overview of the system architecture and your business needs is necessary to identify the functions that should be made ready to be called from the outside.

One more point: coffee is also called Java. Do you need to use Java to enter the world of Web services? No, in this book, we show various other approaches to solve your business requirements.

1.2.2 A non-technical but business-related example

The previous section provided a more general introduction to services. Now, we present an example that is more specifically about services in the IT world.

You are the owner of a general store in town that sells all types of goods. One of your main suppliers is located out of town. Both of you are well equipped with phone lines, Internet access, and computers for your business administration. While you are using a Windows®-based application, your supplier is running business software on a System i platform with applications written in Report Program Generator (RPG), because the business is serving many customers all over the country.

To place an order for new goods with your supplier, you can usually send a fax or an e-mail or you place a phone call. All of these means require an employee of your supplier to enter the
order into their system. This process is not only time-consuming, it is also a source for possible errors due to wrong entries or typographical errors.

Now, if your supplier has a Web site with an order form, you can place that order yourself directly into the system. Assuming that the Web server is running on the System i platform, the supplier still needs to integrate the Web site frontend, possibly written in a combination of HTML and PHP or using JavaServer™ Faces, with the backend software written in RPG.

Let us go back to your side of the business. Suppose that your business software allows you to enter all your sales and keep your stock information up-to-date, therefore allowing you to order new products when stock is decreasing below a certain level. However, you still have to check your stock figures and place that order manually.

Do you see what is coming next? Right: wouldn’t it be nice if your software could—as soon as the stock for a certain product is below a certain level—enter in direct contact with your supplier’s system to order a specified quantity? That automated action would eliminate the need to constantly have an eye on your stock and place an order when needed. On the supplier’s side, it would also eliminate the time-consuming need to enter that order into the system.

Web services can make this type of action happen. In short, a Web service is a function in an application that can be called from a program or a system somewhere else using the Internet, very similar to a Remote Procedure Call (RPC). Instead of a human interacting with a computer through a Web interface, we have applications interacting directly together with the help of Web services.

But wait... we said that one system is running Windows and the other is running i5/OS®? What if instead of Windows, you have a Linux®-based system? The good news is that this process still works. In fact, all big players in the IT industry have joined forces to support a common standard—IBM, Microsoft®, and Sun, to name just a few. This standard makes it possible to integrate systems based on different technologies using the SOA approach.

### 1.3 SOA characteristics

After looking at the concept of services on a general level, followed by a non-technical sample but with some relation to service-based solutions in a business environment, we now have a look at some characteristics that distinguish an SOA from other architectures. With the key components and differentiating points for an SOA, you can:

- Implement SOA in many different ways.
- Take advantage of the benefits of loose coupling.
- Continue to use existing applications.
- Implement quality of services.

#### 1.3.1 Implement SOA in many different ways

SOA is an architecture. As with any architecture, SOA defines the overall design principles for constructing a software component. Implementation of that component, however, is a different thing. Let us illustrate with an analogy about building a house. You can use an excavator to dig the foundation of the house, which will make the job move along quickly and easily, or you can do it using a simple shovel, which will make the task more difficult. Similarly, there might be many ways to implement an SOA solution.

However, most often companies benefit from using a standard or a set of standards for implementing an architecture. SOA is not an exception. There is a set of standards called Web
services. Web services are recognized as the best way for implementing SOA solutions. Web services are developed as an open source project, because they do not lock you into any single vendor. This is one of the main attractive points for adopting Web services.

However, do not assume that SOA and Web services are the same. You can implement an SOA solution using technologies, such as CORBA or .Net, but these technologies do not realize all the possible benefits of SOA. Instead, they imply very strict requirements for implementing a service, possibly locking you into a single vendor.

You can learn more about the Web services standards in Chapter 3, “Web services technology stack” on page 17.

1.3.2 Take advantage of loose coupling

One of the key points of an SOA is that an application’s different functional units, called services, are interrelated through well-defined interfaces and contracts between these services. The interface is defined in a neutral manner that should be independent of the hardware platform, the operating system, and the programming language in which the service is implemented. This independence allows services that are built on a variety of such systems to interact with each other in a uniform and universal manner.

This feature of having a neutral interface definition that is not tied strongly to a particular implementation is known as loose coupling between services. The benefit of a loosely-coupled system is its agility and ability to survive evolutionary changes in the structure and implementation of the internals of each service that make up the whole application. Alternatively, tight-coupling means that the interfaces between the different components of an application are tightly interrelated in function and form, thus making them brittle when any form of change is required to parts of or the whole application.

This benefit of loosely-coupled applications requires an additional functional layer in your solution that has to convert the platform neutral interfaces to the platform specific APIs. However, the benefits of loosely-coupled solutions outweigh the downside of having an additional functional layer.

1.3.3 Continue to use the existing applications

A lot of talk and promises has gone into reusing application code or business logic. Unfortunately, as many have found out, this reuse of existing code is not always as easy as they thought in the beginning. It certainly is more difficult with monolithic applications that might have to be modernized, that is modularized, before reusing parts of its business logic to integrate with other applications or systems. For many, the term reuse remains limited to “change the underlying hardware, recompile, and continue using your applications.”

With an SOA, there is again that promise, but let us say “continue to use the existing” instead of “reusing.” In fact, in earlier times, when two companies with different systems merged, it was often the case that one company took over the system of the other, because integrating them was not possible or too difficult. That type of merger could have a huge impact, especially if the staff on site did not know the operating system and programming languages that was used on the other system.

Now, integrating systems has become easier, because the hardware does not play such an important role anymore. In addition, companies can integrate the software, thanks to SOA, with other systems and, therefore, can continue to use it. Of course, this integration does not come without a cost, but the alternative is not free either.
1.3.4 Implement quality of services

When doing business with an external partner, on whose operations you have no control, you need to define and agree to some rules, often in the form of service-level agreements and operational policies. Understandably, security is very important in enterprise computing. It is equally important that you can trust upon that all business processes are executed reliably and conforming to the terms agreed. This might seem unspectacular for a simple operation but can become very complex in transactions that span over several loosely coupled distributed systems.

Therefore, the following services are part of what is called quality of service:

- Security
- Reliable Messaging
- Transactions

1.4 SOA from a business perspective

One of the goals of IBM is the development and adoption of open standards within its products. SOA is the latest architecture that has received a tremendous interest among CIOs in today's companies. SOA is well positioned to allow business needs to drive development. This positioning enables you to realize the value proposition of SOA within your company's IT infrastructure. SOA promises to optimize the alignment of business needs with IT, decoupling business process activities from service implementations, and to reduce operational costs. You can accomplish these capacities without vendor lock-in, when technologies, targeted for SOA implementations, are integrated seamlessly (open standards) to construct comprehensive end-to-end solutions.

1.4.1 Reasons to consider SOA

On white boards and on paper, the concepts of SOA can be very compelling and more easily justified when the company takes into consideration strategic business goals and initiatives. However, the decision to implement SOA should not be taken lightly. It is similar to committing to a lifestyle change because the IT governance to which your development and operational teams adhere to will be quite different. Business-driven development is one of the key components of SOA. The process involves refining business needs to IT requirements, and then IT requirements to IT capabilities, to identify technology to address the needs. Good reasons to consider using SOA include:

- Your company has existing business logic that needs to be accessible by other intranet applications, strategic business partners, or external Internet applications
- Integration costs continue to grow without being offset by new business opportunities that provide a real return on investment (ROI).
- Mergers and acquisitions are central to your company's business model for growing market share and pursuing new opportunities.
- Solutions require the integration of business capability from disparate systems and programming models.
- The livelihood of your business depends on your ability to adjust quickly to changes in the marketplace or to respond immediately to competitive threats.
- The impact of the global economy necessitates that your company does more with less and that your company relies on business partners to provide non-core business functions.
The efficiency of working with business partners is critical for your company in driving revenue.

The value of your company’s business assets are diminished because they are not assessable for reuse outside their original purpose.

The efficiency of your company’s employees is in question because they do not spend the majority of their time delivering capabilities or services that are core to your company’s business model.

Your company’s business thrives on opportunistic business endeavors.

Your company develops new applications from scratch. Our belief is that SOA should be a default architectural style to position new applications for the future, unless business conditions dictate otherwise.

### 1.4.2 SOA is not always a perfect fit

In an ideal world where there are no budget constraints, schedule deadlines, skill gaps, and priority differences between you and your business partners, it is safe to say that everyone would be adopting SOA or, at least, would have plans to adopt it. However, in the real world, our choices are often influenced and limited by past decisions (for example, investment in technology, adoption of programming models, or commitment to contractual agreements of services). As a result, we are not always at liberty to make what appears to be the perfect choice for addressing a business need or technical requirement. Some indications that SOA might not be a good fit for your company include:

- A small percentage of your company’s IT budget is spent on integration activities.
- A majority of your company’s processes are manual or document-centric, with little opportunity for automation.
- A large majority of your company’s application development uses the same programming model.
- The operation of your company is managed by one or two customer relationship management (CRM) and enterprise resource planning (ERP) applications with little integration requirements.
- There is a significant mismatch between your company’s existing skill base and that which is needed to implement an infrastructure to support SOA.
- A clear business need or opportunity has not been identified that would benefit from the IT capabilities offered by SOA.
- An existing revenue stream would be adversely affected due to the availability of new business services.
- Business partners with whom your company relies upon have different priorities about automating intercompany processes.
- Your company’s primary business revolves around extremely high-volume, synchronous, real-time transactions.

Every engagement or project brings unique requirements, so the decision about whether to adopt SOA depends on your company’s business situation. The value proposition of SOA is hard to trump, but choosing to commit your company to embrace SOA must be balanced by the reality of your business environment. In addition, you do not have to adopt SOA in one large leap. Typically, adoption of SOA is done in small steps. Finding a project where you can use the concepts and principles of SOA and then measuring its value with key performance indicators is powerful in cultivating a community of stakeholders.
1.5 Further reading

If you are interested in knowing more about SOA and the Web services Platform Architecture, see *Understanding SOA with Web services*, by Eric Newcomer, Greg Lomow, ISBN 0-321-18086-0 (Pearson Education).

1.5.1 IBM Redbooks publications

The following IBM Redbooks publications also deal with the subject of SOA and Web services:

- *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461
- *Patterns: SOA Foundation Service Connectivity Scenario*, SG24-7228
- *Patterns: SOA Foundation Service Creation Scenario*, SG24-7240
- *Patterns: SOA Foundation - Business Process Management Scenario*, SG24-7234
- *Enabling SOA Using WebSphere Messaging*, SG24-7163
- *Patterns: SOA Client - Access Integration Solutions*, SG24-6775

1.5.2 Web sites

There are also numerous Web sites that provide interesting material about SOA. Here are just a few:

- SOA introduction
  

- Wikipedia: Service-oriented architecture
  
SOA application design

This chapter discusses the process of designing an application based on SOA. We explain how to identify and design services, which are the building blocks of an SOA application.
2.1 Designing an SOA solution

Designing an SOA application is an art, not a science. In other words, when you understand the key guidelines of SOA, you can design several SOA-compliant application designs. The key architectural principal of SOA is creating loosely-coupled services that can be combined into applications. *Loosely-coupled services* in an application can be used independently of each other. Another way to understand the term *loosely coupled* is by looking at an electrical appliance such as a vacuum. With a vacuum, an electrical cord is tightly coupled with the vacuum but loosely coupled with a power outlet. Loose coupling allows services reuse. In our vacuum example, you can plug any electrical appliance into a power outlet, not just a vacuum, and the vacuum can be plugged in to any power outlet.

A service is a basic building block of an SOA application. The first step in designing an SOA application is identifying and creating services.

2.1.1 Designing services

Few companies today have the luxury of starting software development from scratch. This is especially true for System i customers and independent software vendors (ISVs) who have applications that were developed up to 20 years ago. These applications are robust, stable, and proven. Thus, completely rewriting these applications does not result in a significant return on investment (ROI). At the same time, traditional applications need to be extended to support changing business requirements. Implementing services is an example of extending a traditional System i application.

SOA-based IT infrastructure or a product might be the end goal of any company that started on the SOA path, and creating a service is the first step on that path. What part of your application makes a good candidate for a service? The answer to this question is in *business requirements*. Business requirements can be as informal as one of the users saying “I want to track shipped packages from our Order Entry system” or as formal as “We need to provide standards-based order process for our suppliers.”

**Identifying business functions that make useful services**

In some cases, the process of identifying a service is simplified because services creation is specified as a requirement. For example:

- You need to expose your business functions to external companies, and SOA/Web services was chosen as an architectural approach and a standard.
- Your company requires that all applications provide a standard business logic interface so that applications can be easily integrated into an Enterprise Services Bus or similar infrastructure.

If you do not have an explicit requirement to convert business functions to services, you will need to do some analysis to identify useful services.

Services today are used for functions as simple as a spell check and as complicated as an e-Commerce system. The similarity between these services is that they provide a business function that can be used by multiple applications in multiple contexts.

Here is a sample list of steps for identifying feasibility of a business function as a service:

1. Identify a business requirement.
2. Identify a business task that fulfills the requirement. The business task becomes a service.
3. Verify that there could be at least two different uses (clients) for this service.
Let us apply this decision process to two business requirements in a sample scenario where a company that sells school district management software received two new business requirements from customers. As an example, we show two possible scenarios.

Here is the first scenario:

1. Business requirement: Expose student attendance information to external users.
2. Business task: Look up student attendance in the attendance database and return it in a standards-based format.
3. Verify that there could be at least two different uses (clients) for this service: This service can be used by ISV software and integrated into other systems, such as School District’s Web applications.
4. The ISV made a decision to implement this requirement as a Web service.

Here is the second scenario:

2. Business task: Look up and update teacher’s continuing education credits.
3. Verify that there could be at least two different uses (clients) for this service: At this time, the only use of this service is by ISV software. There is a restricted set of users who can access this information.
4. The ISV made a decision to implement this function as a reusable modular software component but not as a Web service. The deciding factors were only one identified client for this service and a limited group of users.

Why not make every business task a service

One of the things to keep in mind when designing SOA-based applications is implications of loosely-coupled architecture on application complexity, performance, and maintenance. An on-going challenge in software design is the trade off between the best possible architecture and the best architecture for performance. Your task as an SOA architect is to come up with a solution that satisfies both SOA architecture and performance requirements. If performance is a concern, in addition to the decision criteria for identifying services that we outlined in the previous section consider the following criteria:

- Create large-grain services versus creating many small-grain services, that is implement an entire business task as a service versus a component of a business task.
- Provide two interfaces for business logic: the services interface and an interface for integrating with clients written in the same programming language.

Today’s uses of services

Looking at existing services can give you additional ideas on what business functions make good examples of services. For example, United States Postal Service offers address standardization, zip code lookup, and city and state lookup services. Amazon.com, one of the leading online retailers, is a pioneer in providing e-commerce Web services. Using Amazon’s Web services, you can build your own front end to Amazon’s warehouse of products. Another Web service offered by Amazon is a “Simple Storage Service” that provides read/write capabilities for any amount of data over Internet. Other services examples are search and spell check Web services provided by Google.com and services to create maps and driving directions provided by Mapquest.com.
All these examples are services that are created for use by external applications. Services can also be created for use strictly by internal applications. In the intranet environment, services are used to solve a variety of business problems, such as:

- Provide a single point of access to multiple back-end applications
- Integrate applications written in different programming languages
- Build flexible applications that can be easily extended and integrated

An automobile manufacturer, DaimlerChrysler, uses Web services to integrate applications within their human resources, procurement, supply chain management, and manufacturing organizations. Their applications reside on several platforms which include AIX®, OS/390®, SUN Solaris™, and Windows and include Java, COBOL, Visual Basic®, and PowerBuilder applications. Wachovia, one of the leading U. S. financial institutions, uses Web services to integrate Microsoft .Net, the rich client with back-end applications.

You can find additional Web services case studies at:

- IBM Case Studies
- Web Services information

### Service design considerations

After you identify a good candidate for a service, the next task is to design the service interface. The service interface consists of input and output parameters. Your goal should be to design an interface for maximum interoperability. Let us look at the factors that affect interoperability between services and clients implemented in different programming languages:

- **Data type**: Primitive data types (strings, integers, and so forth) are supported by all programming languages. When designing the services interface, use primitive data types and avoid programming language specific data types. If you are planning to use SOAP as a messaging mechanism for services, verify that the chosen data types are supported by SOAP. See Chapter 3, “Web services technology stack” on page 17 for more information.

- **Simple or complex**: A complex parameter includes many simple parameters, usually primitives. For example, we can pass customer information in a complex Customer parameter which contains several attributes—customer name, address, and so forth. In Java, this parameter is implemented as a JavaBean, in RPG as a data structure. We can also pass customer information in an XML document included in a simple String parameter. Which approach is better for interoperability? The simple String parameter provides the best interoperability because strings are supported by most programming languages. The disadvantage of this approach is that the service is not self-describing, that is the service returns a string, but it does not describe the content of the string. In addition, the client has to parse the XML document that is included in the string. With the complex parameter, the service is self-describing: the output is a Customer object that has several attributes. On the client side we do not have to worry about parsing XML because SOAP APIs parse the Customer output parameter and get information back to the client in the form of a Customer data type. The disadvantage of this approach is that not all programming languages (SOAP APIs of these languages) support complex objects. To determine which parameter type to use in your application, create a Web service client prototype in programming languages of your choice and test integration with complex
parameters. Also, consider designing two interfaces for the same service: one with a complex parameter and one with a string that includes an XML document.

If you are creating services over existing code and cannot change method or procedure interface because they are used by other applications, consider building a facade layer that transforms services parameter types to parameters that are used by existing code.

Another decision that you have to make is whether your service is going to be stateless or stateful. While state is usually an implementation discussion, it can have some impact on your service interface. For example, if your service in stateless, and state is maintained on the client side, you might need to pass state information in the input parameter.

Interaction between service clients and services can be synchronous or asynchronous. Asynchronous communication is used when a service takes a long time to complete or when response is not required by the client application. In some cases, you might want to get a confirmation that your request was received by the service but you do not want to wait for results, in other words, a combination of synchronous and asynchronous model. In this case, implement a service that receives a request from a client and posts a message that will later be processed by a batch program. The service can return an output parameter containing request id. In addition, implement other services that will return request status and request results.

You should give special security considerations to services that are created for consumption by external applications. You need to decide if authentication and authorization should be done on a individual service on an application level. If you decide to implement security on a service level, you need to pass user authentication information as an input parameter. If you are implementing security on an application level, you might still need to pass a parameter to each service verifying that the user has been authenticated.

Creating SOA applications from services

After you create services, you need to integrate them into existing or new applications. If you are adding services to an existing application create a services layer or an adapter that encapsulates access to all services. Figure 2-1 illustrates a sample application architecture.

![Sample application architecture](image)

You should give special security considerations to services that are created for consumption by external applications. You need to decide if authentication and authorization should be done on a individual service on an application level. If you decide to implement security on a service level, you need to pass user authentication information as an input parameter. If you are implementing security on an application level, you might still need to pass a parameter to each service verifying that the user has been authenticated.

Creating SOA applications from services

After you create services, you need to integrate them into existing or new applications. If you are adding services to an existing application create a services layer or an adapter that encapsulates access to all services. Figure 2-1 illustrates a sample application architecture.

![Sample application architecture](image)

If you are building new applications based on a large number of services, consider using Business Process Execution Language (BPEL) to define and create composite services and process choreography to define relationships between services. Both technologies are industry standards, WS-BPEL and WS-CDL (Choreography Description Language),
respectively. IBM provides tools and middleware for building SOA applications using BPEL and process choreography.

See the following Web sites for more information and links about BPEL:

- Wikipedia: Business Process Execution Language
  http://en.wikipedia.org/wiki/BPEL

- Web Services Choreography Description Language Version 1.0
  http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/
Web services technology stack

This chapter describes technologies that are used for building and invoking Web services. We describe the following technologies in this chapter:

- SOAP
- HTTP
- WSDL
- UDDI
- Basic Profile
3.1 Web services technologies in action

Web services technologies are best explained in the context of how they are used during the development process and at runtime. A developer implements modular code, for example an RPG module with several procedures or a JavaBean with several method according to a previously defined interface. At this point, no Web services-specific code has been implemented. Next, the developer creates a Web service from modular code. Web services creation does not add logic to implemented modular code, but creates the infrastructure around it that allows modular code to be invoked as a Web service. Most IDEs, including WebSphere Development Studio client for iSeries®, provide tools for creating Web services from existing code. One of the key artifacts generated during Web services creation is Web services Definition Language (WSDL) file. WSDL is an XML document that describes Web service interface: methods, parameters and service location.

The last step in developing a Web service is to test it. Communication between Web service clients and Web services is achieved with SOAP—an XML standard for the exchange of structured information in a distributed environment. A test application, or Web services test client, must create a SOAP message to invoke a Web service. You can create test client manually or use tooling in WebSphere Development Studio client for iSeries to generate test client code.

Web service client developer needs information about the Web service to implement Web service invocation code: service location, methods, and parameters. As mentioned previously, this information is captured in the WSDL file. Using information in the WSDL file, the Web service client developer uses SOAP API to create a SOAP request message to invoke the Web service. WebSphere Development Studio client for iSeries and some other IDEs provide wizards to generate SOAP APIs based on a WSDL document. A term Web services invocation proxy or just proxy is used usually to describe a generated code artifacts that includes SOAP API.

Next, let us take a look at how Web services technologies are used at runtime. A Web service client creates a SOAP message and sends it over HTTP or another transport protocol to a known service location. If HTTP is used, the service location is specified as a Uniform Resource Identifier (URI), for example http://myhost:9080/SampleWebService/services/SampleWS. Web service is deployed in an application server with a special component, SOAP server, that is responsible for processing SOAP messages. The SOAP server processes the incoming message, invokes the Web service, and sends the response message back to the client. Some implementations of SOAP server use WSDL to process request and response messages.
Figure 3-1 shows the interaction between the Web service and the Web service client.

With the right tooling, for example, WebSphere Development Studio client for iSeries or IBM Rational tools, Web service and Web service client developers might not have to write any Web services-specific code. However, it is still important to understand some basics about Web services technologies.

Web services technology stack consists of industry standard technologies that can be divided into four categories:

- Web services messaging
- Web services transport
- Web services description
- Web services discovery

### 3.2 SOAP: Web services messaging layer

From the client perspective, a Web service call is a remote procedure call (RPC), and one of the main differences between Web services and other inter-program communication mechanisms is the way that a program is invoked. SOAP is an XML dictionary that's used to perform a remote procedure call. A more generic definition of SOAP is a specification for the exchange of structured information in a decentralized, distributed environment. Key characteristics of SOAP are:

- SOAP is transport-independent protocol and can be used in combination with a variety of protocols such as HTTP, JMS, SMTP, or FTP. Today, the most common way of exchanging SOAP messages is through HTTP
- Because SOAP is an XML dictionary any programming language that can process XML can create SOAP messages
- A SOAP message is an envelope that includes zero or more headers and one body.
  - The envelope is the top element of the XML document, providing a container for control information, the addressee of a message, and the message itself.
  - Headers that include control information, such as quality of service attributes.
  - The body includes the message identification and its parameters.
Let us take a look at a simple Web service and a SOAP message that is used to invoke it. Example 3-1 shows Java Web service code. Notice that there is no Web service specific code in the method implementation because Web services technologies are used outside of the business logic implementation.

**Example 3-1 A business logic method**

```java
public String convertTemp(String tempIn) {
    Double tempFahrenheit = new Double(tempIn);
    // Convert temperature
    Double tempCelsius = new Double ((5/9)*(tempFahrenheit.doubleValue()-32));
    // Return a String value
    return tempCelsius.toString();
}
```

Example 3-2 shows SOAP request message from the Web service client to invoke this Web service. Notice that the SOAP request message includes the name of the service to call (which matches the Java method name), and the input parameter `tempIn` with a value of 90.

**Example 3-2 Sample SOAP request**

```xml
<soapenv:Envelope 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>
    <p831:convertTemp xmlns:p831="http://services.ibm.com">
      <tempIn>90</tempIn>
    </p831:convertTemp>
  </soapenv:Body>
</soapenv:Envelope>
```

Example 3-3 shows the SOAP response message. The most interesting part of the message is the return parameter, `convertTempReturn`, that includes the converted temperature.

**Example 3-3 Sample SOAP response message**

```xml
<soapenv:Envelope 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>
    <p831:convertTempResponse xmlns:p831="http://services.ibm.com">
      <convertTempReturn>32.2</convertTempReturn>
    </p831:convertTempResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

If you do not have any SOAP code in the Web service implementation, how are the SOAP request and response messages created? The SOAP request message is created by the Web service client. Web service client can use SOAP APIs of the programming language in which the client is written. Most programming languages have a SOAP API. If they do not, it is possible to create a SOAP message manually, but this approach can be tedious and error-prone. If the programming language does not have a SOAP API, consider building a SOAP client proxy in a programming language that has a SOAP API. Many IDEs, including
WebSphere Development Studio client for iSeries, can generate Web service client code that includes SOAP APIs based on a WSDL file.

Web services response message can be generated by the SOAP engine or created manually with the SOAP API. The SOAP engine also handles SOAP request messages. WebSphere Application Server comes with a built-in SOAP engine. In addition, you can use the open source Axis SOAP engine in WebSphere Application Server. WebSphere Web services engine is based on Axis principles but extended for performance and for enterprise Web services (support for session EJB as Web services and for SOAP over JMS).

3.3 Web services transport

SOAP messages can be sent over several protocols, including HTTP, JMS, FTP, and SMTP. HTTP and messaging protocols, for example JMS (Java Messaging Service), are currently the most popular choices for Web services transport. HTTP provides synchronous communication between a Web service client and a Web service. HTTP is the most used protocol for several reasons:

- HTTP is mature: It has been used for Web applications for several years.
- HTTP requests can travel through firewalls. (HTTP ports are already configured for Web applications.)
- HTTP can be secured with SSL.
- HTTP is the only protocol supported by the basic profile standard (which we explain in 3.6, “Basic Profile” on page 24).
- Most IDEs, including WebSphere Development Studio client for iSeries Web services wizard, generate Web services infrastructure for HTTP communication.

JMS provides asynchronous communication between a Web service client and a Web service:

1. The client request to the Java proxy is handled by the SOAP client and is placed into a JMS queue through a JMS sender.
2. In the server, a message-driven EJB (MDB) listens to the JMS queue and routes the message to the WebSphere SOAP engine.
3. The WebSphere Web services engine invokes an EJB Web service.
4. Optionally, the server replies to the client using a dynamic queue.

WebSphere Development Studio client for iSeries does not yet provide tooling to generate Web services infrastructure for JMS transport. This functionality has to be implemented manually. For more information about JMS implementation, see Web Services Handbook for WebSphere Application Server 6.1, SG24-7257.
3.4 Web services description: WSDL

A WSDL document describes a Web service: Web service location, methods, and method parameters. Example 3-4 shows a sample JavaBean implementation. Notice that this Web service has one method `convertTemp` that has an input String parameter and output String parameter.

```
Example 3-4  Sample JavaBean implementation

public class TempConversionService {

    public String convertTemp(String tempIn){

        Double tempFahrenheit = new Double(tempIn);
        // Convert temperature
        Double tempCelsius = new Double ((5/9)*(tempFahrenheit.doubleValue()-32));
        // Return a String value
        return tempCelsius.toString();
    }
}
```

Example 3-5 shows a WSDL file that the WebSphere Development Studio client for iSeries Web services wizard generated for this JavaBean. While the document can seem complicated at a first glance, you can still see familiar elements from the Web service implementation: Web service method name and parameter definitions. At the end of the document you find the following Web services location:

```
Example 3-5  Sample WSDL file

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://services.ibm.com"
xmlns:impl="http://services.ibm.com" xmlns:intf="http://services.ibm.com"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/
xmlns:wsi="http://ws-i.org/profiles/basic/1.1/xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <wsdl:types>
        <schema targetNamespace="http://services.ibm.com"
xmlns="http://www.w3.org/2001/XMLSchema" xmlns:impl="http://services.ibm.com"
xmlns:intf="http://services.ibm.com"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <element name="convertTempResponse">
                <complexType>
                    <sequence>
                        <element name="convertTempReturn" nillable="true" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
        </schema>
    </wsdl:types>
</wsdl:definitions>
```

Notice that the generated URL points to `localhost`. This URL is the only part of the WSDL document that you need to change before deploying the Web service to the server and sending WSDL to the Web service client developer. You need to replace `localhost` and the default port with the host name of your server and WebSphere Application Server port.
The WSDL file is sometimes called a contract between a Web service and a Web service client. After you create a contract, your goal should be not to introduce any changes to it. Otherwise, the communication between a Web service client and a Web service will be broken. You can change Web service implementation (for example, change the way you calculate temperature conversion). However, if you change your method of signature (method name or parameter types), you will break the contract. Changing Web service location will not break the contract. Just remember to notify the Web service client of Web service URI change.
Some developers prefer to start Web services development by manually creating the WSDL file and then generating a programmatic interface (JavaBean or another code artifact usually called a skeleton or a stub) based on the WSDL file. This approach requires advanced XML and WSDL skills. You can learn more about WSDL in Web Services Handbook for WebSphere Application Server 6.1, SG24-7257.

### 3.5 Web services discovery: UDDI

In general, SOA overview Web services discovery is often described as a part of interaction between a Web service client and a Web service. *Universal Description, Discovery, and Integration* (UDDI) is the component used for Web services discovery. UDDI is also called yellow pages or a broker, because it includes information about Web services. Web service developers publish information about their Web services to UDDI (usually a WSDL file). A Web service client can lookup this information at either design or run time.

Figure 3-2 shows interaction between a Web service client, a Web service and UDDI.

![Figure 3-2  UDDI interaction](image)

While UDDI has been introduced in the very early descriptions of SOA, it is not widely used in Web services implementations today. Most Web service client and Web services developers exchange Web service description information without UDDI. UDDI might become more popular when many vendors deploy similar services (for example, a credit card check), and a UDDI provider decides to become a broker of these services.

Another way to describe a set of Web services is Web services Inspection Language (WSIL). WSIL is not a repository like UDDI, but an XML file that describes how to find Web service.

### 3.6 Basic Profile

One of the main goals of Web services is to provide interoperability between applications written in different programming languages using different IDEs and running on different middleware. However, using Web services does not mean that your application is going to integrate seamlessly with any Web service client. The key to interoperability is using the same version of core Web services technologies: SOAP, WSDL, HTTP, and others. That is why the Web services Interoperability Organization created Basic Profile, which is a set of guidelines for creating interoperable services.
Basic Profile V1.0 specifications include:

- SOAP V1.1
- WSDL V1.1
- UDDI V2.0
- XML V1.0 (Second Edition)
- XML Schema Part 1: Structures
- XML Schema Part 2: Datatypes
- RFC2246: The Transport Layer Security (TLS) Protocol V1.0
- RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile
- RFC2616: HyperText Transfer Protocol V1.1
- RFC2818: HTTP over TLS
- RFC2965: HTTP State Management Mechanism
- The Secure Sockets Layer (SSL) Protocol V3.0

The profile adds constraints and clarifications to those base specifications with the intent to promote interoperability. Some of the key constraints include:

- Precludes the use of SOAP encoding (document/literal or RPC/literal must be used)
- Requires the use of SOAP/HTTP binding
- Requires the use of HTTP 500 status response for SOAP fault messages
- Requires the use of HTTP POST method
- Requires the use of WSDL V1.1 to describe the interface
- Precludes the use of solicit-response and notification-style operations
- Requires the use of WSDL V1.1 descriptions

Basic Profile specification is used mostly by tool and middleware developers. They need to make sure that their products comply with the Basic Profile specification so that Web services can interoperate. As a Web service or a Web service client developer, you need to be aware of the Basic Profile version supported by technologies that you use. WebSphere Development Studio client for iSeries V6.0 and later and WebSphere Application Server V6 and later support Basic Profile V1.0. To ensure interoperability between a Web service and a Web service client, make sure that they support the same version of the Basic Profile.

### 3.7 Summary

In this chapter we highlighted the most important Web services technology standards. For a full description of Web services standards, see *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257.
Sample scenario

In this chapter we introduce the sample scenario used throughout this book to provide a common basis of understanding. We describe a fictitious travel agency and its current system. From there we provide ideas to solve some of the issues the company is facing. Later on in this book we show several solutions to these issues.
4.1 Introducing the existing environment

In our scenario, we use the sample application called Flight400 that has been used in other IBM Redbooks and presentations. The Flight Reservation System application is representative of a commercial application. Though the application does not include all of the necessary error handling that a typical business application requires, it has the logic that we use to demonstrate how we can modernize applications and make parts of it available through Web services.

4.1.1 Overview of the Flight400 application

The Flight Reservation System application is used by the agents to create, query, modify, and print reservations. For example, in a case of a reservation received, an agent is required to manually enter necessary data into the system.

For more information about the Flight400 application and articles on various topics around that sample application, check out the following Web site:

http://www-03.ibm.com/servers-enable/site/ideveloper_j2ee/etoe/index.html

4.2 Facing new challenges

As everywhere, a business is changing and customers are expecting more or better service (or actually both). To stay ahead of the competition, we constantly have to think about ways to improve our offer.

4.2.1 New opportunities

Currently, creating a new reservation requires a lot of manual work. But let us step back and look at the business process that triggers this action.

An employee of one of our partner companies, a travel agency, desires to make a new reservation on behalf of one of their clients. She picks up the phone and calls our office. We get the call, start the Flight400 application, select the option to make a new reservation and then enter all the necessary data that we get from our partner over the phone.

Problems might arise if we misunderstand our business partner or mistype the information received. If the information is received by letter, fax or by e-mail, we still have to transfer that information into the system in a manual process, which is both time-consuming and error-prone. But the most time-consuming part is to provide information about flights and customer data over the phone.

Therefore, we identify these business cases as the first ones that we want to modernize and possibly automate. The idea being that our business partners can query information about flights or customers themselves without interaction from our part and enter their reservations directly into our system.
4.2.2 Technical issues to address

However, there are some important points that we need to consider:

- Some of our partners are running on different platforms and are possibly using different Database Management Systems (DBMS).

  The solution to this possible issue in our case is to implement the identified functions as Web services and therefore allow their functionality to be exposed to the outside world, which in this case are our business partners.

  But this brings up another important point that needs to be considered even before thinking about connecting to the outside world:

- Our application might have to be modularized, respectively modernized before we can re-use its functions in the form of Web services.

  In fact, RPG programs can be converted to Web services by building a Java Web service wrapper around them, either automatically or manually. But before that step, you need to make sure that the RPG program implementation is suitable for a Web service. The following two main requirements must be met:

  - Separation of business logic and presentation logic. The RPG program must be a callable program that does not contain any display logic.
  
  - Thread safety. By default RPG programs are not thread safe. If two or more Java threads call the same RPG program at the same time, you might get unexpected results.

For more information about RPG application modernization, read Modernizing Flight400 white paper at:

http://www-03.ibm.com/servers/enable/site/education/wp/40d2/40d2.pdf

Implementing thread safety in an RPG program

Programs or routines need to be thread-safe in order to avoid the risk that one thread interferes or even modifies data elements used by another thread. In a Web service context that could be the same RPG program which is being called from two or more different processes where each process generates a thread calling the same RPG program. It could be fatal if this program modifies some global data or the heap.

- RPG IV has a thread safety option that can be specified on the H spec: THREAD(*SERIALIZE). When the program is compiled with this option only one thread will be active in one module at one point in time. However, you still have to make sure that shared storage (such as IMPORT/EXPORT fields) is handled in a thread-safe way.

- The RPG IV runtime is thread-safe, RPG II and RPG III are not thread-safe.

- It is not possible to have thread-scoped files using RPG file support. If the file is left open, the next thread will get the old file state. On the other hand, it is possible to have some of the storage thread-scoped by using allocated storage or userspaces. The program will have to know which thread is active (using a parameter from the caller) to know which basing pointer or userspace to use.

- C runtime I/O functions should be used to do thread-scoped I/O, if necessary.

We highly recommend to read the text on thread-safety on Wikipedia, as it provides some basic but important definitions and outlines ways to achieve thread-safety in programs. There are also links to articles on how to write thread-safe programs:

http://en.wikipedia.org/wiki/Thread_safety
4.2.3 Flight400: From monolithic System i application to SOA

In the previous sections we demonstrated the valid justifications for creating Web services. In this section we elaborate on the previous example to identify other parts of the application that are the good candidates for Web services.

The original version of Flight400 application is a flight reservations system written in RPG III application. Our business requirement is to allow other applications (internal or external) place reservations in the Flight400 reservation system.

We started the design process with identifying tasks in our typical daily use of the application:
1. Lookup flights based on a search criteria
2. Lookup to/from cities
3. Lookup airlines
4. Get detailed flight information
5. Place reservation
6. Update reservation
7. Delete reservation
8. Add customer
9. Lookup customer

Each task might or might not be a good candidate for a service. For example, services 2 and 3 (cities lookup and airline lookup) could be considered too granular and useful only within the application, and not as an external services. We decide not to implement these services. You should do these analysis for all tasks.

Tip: If you're just starting with Web services, pick one task that is self-contained. It will be the prototype to understand time and effort involved in developing Web services.

Next, we defined input and output parameters and parameter types for each valid service. For example, the flight lookup service has four input parameters: from city, to city, departure date, and return date. The output of the lookup service is a list of flights that match the search criteria. We decide to use both simple and complex parameters for services. Flight lookup service has four string input parameters and returns multiple occurrences of the Flight data structure. All of our services are stateless and synchronous, state is maintained on the client.

The next step is to review existing RPG code and to find pieces of the code that can be used in the service. We create a new RPG ILE prototype and a module that contains a procedure with input and output parameters identified in the previous step.
Example 4-1 shows the prototype of RPG procedure. The prototype describes input and output parameters for three RPG procedures and a data structure that represents flight information.

**Example 4-1  RPG procedure prototypes**

```rpg
   d FlightInfo      ds                  qualified
   d  Airline                       3
   d  Flight                        7
   d  DoW                           2
   d  DepartCity                    3
   d  ArriveCity                    3
   d  DepartTime                    8
   d  ArriveTime                    8
   d  Price                         3

   d FindFlightsDoW  pr
   d   FromCity                    16    const
   d   ToCity                      16    const
   d   DeptDoW                     16    const
   d   ReturnDoW                   16    const
   d   FlightCount                 10i 0
   d   Flights                           likeds(FlightInfo) dim(50)

   d FindFlights     pr
   d   FromCity                    16    const
   d   ToCity                      16    const
   d   DeptDate                     8    const
   d   ReturnDate                   8    const
   d   FlightCount                 10i 0
   d   Flights                           likeds(FlightInfo) dim(50)

   d GetFlightInfo   pr
   d   FlightNumber                  7    const
   d   FlightInfo                         likeds(FlightInfo)
```
After designing services interfaces, we identified workflow between the client and services. In our scenario services are used independently of each other, but they need to be executed in a particular order. Figure 4-1 shows Flight400 application workflow.

![Application workflow](image)

**Figure 4-1  Application workflow**

### 4.3 Conclusion

As the result of this design effort, we have identified the services that we want to build. We have prototyped the procedure calls in RPG. The procedure calls should match our Web services interface.

The next step is to change, if needed, the business logic that implements the services. In our scenario the original RPG application requires additional effort to make it modular (according to the ILE concepts). You can find the detailed instructions for modernizing the Flight400 application at:

http://www-03.ibm.com/servers/enable/site/education/wp/40d2/40d2.pdf

After we have the callable interface to the RPG application that matches the Web services interface, we need to implement a Web service itself. This includes:

- Implementing a Web service. It will act as a "glue" layer between platform independent SOAP messages and platform specific business logic. The Web service is responsible for:
  - Receiving and unpacking the SOAP request message
  - Invoking the business logic with the correct arguments
  - Returning a result of the business logic execution, if appropriate, in a SOAP response message
- Creating a client for testing a Web service
- Deploying a Web service along with modified business logic on a production server

All these steps are discussed in the next chapters.
Part 2

Implementing the service provider

SOA defines two major participants in its architecture: a service provider and a service consumer. In this part of the book, we demonstrate several technologies that you can use to create a service provider participant. This service provider builds Web services around i5/OS applications.

Note: We developed all examples in this book using WebSphere Development Studio Client for iSeries V6.0.1.

We use Flight400 RPG application for most examples in this book. You can download this application using the instructions in Appendix D, “Additional material” on page 281.
ProgramCall (RPG, Cobol) Web service

Our first example of building a Web service is based on the ProgramCall capability in IBM Toolbox for Java (5722-JC1). In this chapter we discuss the typical method for generating a Web service that invokes an RPG procedure.
5.1 Project investments in developing a service

In externalizing existing business logic as a Program Call Web service, you should consider the following items:

- Analysis of the current state of a business application
- Time frame for prototyping and deploying a services application
- Application development and deployment expenses

5.1.1 Analyzing the existing application

How do you determine whether your business logic needs modernization? The term *monolithic* has been used to describe an application that requires modernization. In a monolithic application, there are two primary areas for analysis:

- Display logic
- Essentially stateless procedure

**Important:** You do not have to rewrite an entire RPG business application to take advantage and demonstrate the power of Web services for the RPG business logic.

Displaying logic nested within business logic

Many System i applications are written as a monolithic application where display and business logic are intermixed. As the result, there are no callable interfaces to such applications.

Consider a company with a monolithic application. Can you still take advantage of the Web services framework without a complete rewrite of the application? The answer is yes. Because a business task was described as service, you can extract and create a procedure that does not include display logic but still includes the business logic with the required input and output parameters.

The whole process involves several steps. Start with a single business task that is self-contained (that is, it does not depend on other tasks). Extract the business logic into an RPG procedure and create a service program. Use WebSphere Development Studio client for iSeries to generate a Web service.

Essentially stateless procedure

*Essentially stateless procedure* is the term for a procedure or a set of procedures that can be used in stateless environments such as Web applications. While some state might be kept between procedure calls (for things like list processing) within a transaction, the procedures should not maintain state between transactions.

Application modernization

Traditional (*green screen*) applications are often written as monolithic programs with a display and business logic intermixed. While there are tools and wizards that you can use to generate Web services from callable routines, a process of converting a monolithic application to a set of callable routines requires a significant amount of design and programming effort.

The steps required to create a set of callable routines from an existing application include:

1. Determine what functions are required. The functions should implement business operations that are required by the Web service interfaces.
2. Design interfaces for the required functions. The interfaces should be essentially stateless.
3. Determine where those functions are currently implemented.
4. Decide how much of the current implementation should be used as a base for the new functions.
5. Develop the new functions.
6. Determine whether the new functions should be used in the existing (green screen) application.

The callable functions should be essentially stateless because the application state is controlled by the calling applications and to allow the use of techniques like connection pooling which are critical for scalable Web applications. This requirement drives several new considerations for the function developer, particularly in list handling and record locking.

Traditional applications handle lists using the display file subfile support. Because a subfile can handle multiple records, the application programmer does not have to be particularly concerned about the number of records in a list. Because the application maintains the data file cursor position between user interactions, the application programmer does not have to be particularly concerned about cursor positioning as the user scrolls through a list. The introduction of essentially stateless functional interfaces requires that a programmer handle these considerations in the function code (on the service requestor side).

There are several techniques that can be used to handle a variable length list as set of callable functions. The most flexible technique is to use multiple functions, the first function sets up the request, the second function returns the next record in the list, and the third (and optional) function closes the request. Because this technique can potentially result in a large number of functions calls, you should not use it in cases where there is a large call overhead. Another technique is to return a set of records as a parameter of the function call. The main disadvantage of this technique is that the parameter has to be defined large enough to handle the maximum expected number of records. A practical approach is to use the first approach as the lowest level interface and use those functions in a second level function that can be used where call overhead is a concern. There are also techniques using SQL result sets or messaging interfaces that can provide more flexibility in distributed applications.

The second consideration in list handling is list continuation between user interactions. List continuation is not usually a problem if the data includes unique keys. However, it does require application support, which can be difficult in data without such keys.

The stateless nature of Web interaction can make record locking a concern. Because a user can leave the application or shut down the browser without notifying the application, locking a record between user interactions can cause problems. One approach is to use some type of optimistic locking, where the application sends both the before and after record images to a function, which does the update only if the before image matches the data currently in the record. While this can work when the amount of data being updated is relatively small, it can also result in problems if the amount of data is large and a user is told that all the changes must be entered again.

While converting a monolithic application into a modular set of callable functions can be a major effort, it can extend the life of the application and make it available through a wide variety of interfaces. In addition, it can result in an application that is more easily maintained and expanded.
5.1.2 Time frame

In this section, we analyze how much time should be allocated to use the program call beans interface into an RPG business logic.

Prototype
Identify a single business task and ensure that an RPG procedure is modularized as described in 5.1.1, “Analyzing the existing application” on page 36.

**Important:** You should not have to rewrite the entire RPG business application to take advantage and demonstrate the power of Web services for the RPG business logic.

Using WebSphere Development Studio client for iSeries, the Web service and Web service client can be prototyped within a few hours. Using the example in this chapter as a reference, any exported RPG procedure can be prototyped rather quickly. WebSphere Development Studio Client generates everything from Web service to JSP and Web service Test Client, directly from RPG source code.

Production
After prototyping is done, is the service production ready? There are additional considerations to keep in mind. Additional testing of a service is essential for the business logic and business application to continue to run smoothly.

In addition to testing the application with a variety of clients that might interface with your service, it is also advisable to ramp up the number of users that might be accessing this business logic to test the throughput. There are products such as Rational Performance Tester or Mercury LoadRunner that would simulate multiple users accessing your application.

Security is a very big concern if you need to pass sensitive information, such as a credit card number. Web services provides several standards that deal with this issue. There is a specification called WS-Security built into the Web services framework. WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. You can use these mechanisms to accommodate a wide variety of security models and encryption technologies.

5.1.3 Development environment

To develop a Web service based on ProgramCall successfully, ensure that your environment meets the requirements that we provide in this section.

Prerequisites

**Note:** We did the testing in this chapter on i5/OS Version V5R4, V5R3, and V5R2. Other versions of the operating system should work too.
Ensure that the following products and options are already present in the environment:

- Qshell, 5722SS1 Option 30
- Host Servers, 5722SS1 Option 12
- IBM Developer Kit for Java, 5722JV1 Options 5 and 6
- IBM Toolbox for Java, 5722JC1
- 5722WDS Option 31 Compiler - ILE RPG IV
- 5722WDS Option 34 Compiler - RPG/400®
- The latest Group PTFs are installed on the target System i:
  - Cumulative Group PTF SF99540
  - WebSphere Group PTF SF99312
- The Host Servers are running on the target System i:
  STRHOSTSVR *ALL

The following products are installed on the development machine (such as a desktop PC):

- WebSphere Development Studio Client for iSeries V6.0.1. Use Rational Product Updater to install the latest fixes for V6.0.1
- Firefox 1.0.7 or higher (or equivalent browser).

**Skills assumptions**

This technical reference assumes the following technical skills and knowledge:

- Experience with and basic working knowledge of tools such as Rational Web Developer or WebSphere Development Studio client for iSeries
- Basic knowledge of i5/OS administration
- Familiarity with the behavior and operation of the existing application that you want to modernize

**5.1.4 Deployment environment**

The deployment environment should include:

- WebSphere Application Server - Express, Base, or Network Deployment running on one of the supported platforms. We run it on i5/OS.
- Server on which the RPG application is deployed.

In addition, you should have the following skills:

- System i administrator
- WebSphere Application Server administrator
5.2 ProgramCall bean example

In this example we demonstrate the RPG business task in Flight400 called ReserveFlight. We use WebSphere Development Studio client for iSeries Advanced Edition to create a Web service from RPG source code using single wizard and publish it to i5/OS using WebSphere Administrative Console, as illustrated in Figure 5-1.

![Web service diagram](image)

Figure 5-1 Web service diagram

5.2.1 Starting WebSphere Development Studio client for iSeries

Open WebSphere Development Studio client for iSeries and set up the workspace by following these steps:

1. Go to Start → All Programs → IBM Rational → IBM WebSphere Development Studio Client for iSeries V6.0 → WebSphere Development Studio Client for iSeries.

2. In the Workplace™ Launcher window, in the Workspace field, enter c:\temp\redbook and click OK as shown in Figure 5-2.

![Workspace Launcher](image)

Figure 5-2 Start WebSphere Development Studio Client for new workspace
3. If you did not use WebSphere Development Studio client for iSeries before, you see a Welcome page. Click X to close the Welcome page, as shown in Figure 5-3.

![Figure 5-3 Close the Welcome Page](image)

**Note:** There are two known issues with running WebSphere Development Studio client for iSeries. If you encounter any problem, see Appendix A, “Setting the connection to WebSphere Application Server V6.0” on page 249 and Appendix B, “URI length limit of 259 characters on Windows” on page 255 for possible solutions.

### 5.2.2 Opening Remote System Explorer perspective

Open the perspective for System i development:

1. Click **Open perspective icon** on the right-hand side or select **Window → Open Perspective** and select **Other** (see Figure 5-4).

![Figure 5-4 Open Remote System Explorer Perspective](image)

2. In the Select Perspective window, select **Remote System Explorer** and click **OK**.

### 5.2.3 Defining a connection to the i5/OS server

Create a connection to your i5/OS server:

1. Click the plus sign (+) next to **iSeries** under New connection in Remote Systems view.
2. If you have not used WebSphere Development Studio client for iSeries before, you see name personal profile screen. Enter redbook as the Profile and click **Next**.
3. On the Define connection information screen, enter the following information (as shown in Figure 5-5):
   a. Host name as <iSeries_Server>.
   b. Enter Connection name as <iSeries_Server>.

![Figure 5-5  Define connection to System i](image)

4. Click **Finish**.
5. Right-click **iSeries Objects** under itso and select **Connect** to connect to the i5/OS server, as shown in Figure 5-6.

![Figure 5-6 Connect to System i](image)

6. In the Enter Password window, enter the following information and click **OK** (see Figure 5-7):
   a. User ID as your user profile on the system.
   b. Password as password for the user profile entered in the previous step.
   c. Select **Save user ID**.
   d. Select **Save password**.

![Figure 5-7 Connection settings](image)
7. After successful connection you should see an arrow next to iSeries connection as well as the iSeries Objects and other subsystems.

5.2.4 Reviewing the RPG modules

Examine the RPG code to understand exported procedures:
1. Expand your System i connection.
2. Right-click iSeries Objects and select Properties.
3. In the Properties window, click Initial Library List.
4. Enter FLGHT400 in the Library field and click Add(B) as shown in Figure 5-8.

![Figure 5-8 Add Library](image)
5. Add the FLGHT400M library in the same way. Your library list should look similar to that shown in Figure 5-9.

![Library list](image)

Figure 5-9  Library list

6. Click OK. Now every time you reconnect to your System i platform, both libraries are listed under iSeries Objects → Library list. You might need to disconnect and connect again to see these libraries in the list.
7. Expand **iSeries Objects** → **Library list** → **FLGHT400M.*lib.prod-usr** → **QRPGLESRC** and double-click **NFS001.rpgle**, as shown in Figure 5-10.

![Figure 5-10 Open RPG Source for ReserveFlight](image)

8. The procedure that we are interested in is **ReserveFlight**. Click the refresh button in the outline frame as shown in Figure 5-11.

![Figure 5-11 Refresh RPG Code](image)
9. Select **ReserveFlight** from the outline view as displayed in Figure 5-12.

![ReserveFlight Procedure](image)

**Figure 5-12 ReserveFlight Procedure**

10. Review program code for the ReserveFlight procedure as shown in Figure 5-13.

![RPG ReserveFlight](image)

**Figure 5-13 RPG ReserveFlight**
The ReserveFlight procedure inputs OrderInfo, which is a data structure of ReserveInfo. The ReserveFlight returns a confirmation number with OrderNumber displayed (see Figure 5-14).

```
00121  d ReserveInfo  ds qualified
00122
00123
00124  d AgentNumber  98  0
00125  d CustNumber   95  0
00126  d FlightNumber 7
00127  d DepartDate   6
00128  d DepartTime   6
00129  d Ticket      3  0
00130  d ServiceClass 1
```

*Figure 5-14 OrderInfo is data structure with following declarations*

11. Click X in the NFS001.RPGLE tab at the top of the window to close the window.

### 5.2.5 Creating and testing RPG Web service

In the previous section, you examined the RPG procedures. Now, you create a Web service directly from the RPG source code that externalizes the RPG procedure:

1. Under QRPGLESRC, right-click **NFS001.rpgle** and select Web services → Create Web service as shown in Figure 5-15.

*Figure 5-15 Create Web service from RPG code*
2. In the Web service dialog box, make sure that the following options are selected as shown in Figure 5-16:
   - Web service Type: iSeries Program Web service
   - Check Start Web service in Web Project
   - Check Generate a Proxy
   - Check Test the Web service

3. Click Next.
4. On the Object Selection Page, do the following (see Figure 5-17):
   - Select **COMPUTEPRICE** in Program Call Definition and change the value of Program Object to NFS001.
   - Select **FINDORDERCUST** in Program Call Definition and change the value of Program Object to NFS001.
   - Select **FINDORDERDATE** in Program Call Definition and change the value of Program Object to NFS001.
   - Select **GETORDERINFO** in Program Call Definition and change the value of Program Object to NFS001.
   - Select **RESERVEFLIGHT** in Program Call Definition and change the value of Program Object to NFS001.
   - Select **UPDATEORDER** in Program Call Definition and change the value of Program Object to NFS001.

Figure 5-17  Setting Service Program for Program Object
5. Expand **RESERVEFLIGHT** and select the **ORDERNUMBER** and change the usage value from input & output to **output** as shown in Figure 5-18.

![Figure 5-18 Setting ReserveFlight Parameter](image-url)
6. Click **Edit** as in Figure 5-19.
7. Ensure all the System i information is correct (see Figure 5-20) and click **Next**. Remember that you might have a different system name and user ID.

![Configure Authentication](image)

Figure 5-20  System i Connection Values
8. Ensure FLGHT400 and FLGHT400M are on the library list as in Figure 5-21. If they are not, enter FLIGHT400 in the Library field and click Add button. Repeat this step for the FLGHT400M library.

9. Click Finish.
10. Ensure that **RESERVEFLIGHT** is highlighted or selected and click **Next** (see Figure 5-22).

![Object Selection Page]

**Figure 5-22** Reserve Flight is selected

11. On Service Deployment Configuration frame click **Next**. This step might take some time to complete.

12. On Service Endpoint Interface frame click **Next**.
13. On Web service Java Bean Identity frame:

   a. Click **Deselect All**.
   
   b. Select `reserveflight(iseries.wsbeans.reserveflight.RESERVEFLIGHTInput)`.
   
   c. Scroll down and select `reserveflight_XML(iseries.wsbeans.reserveflight.RESERVEFLIGHTInput)` as shown in Figure 5-23.

![Web Service Java Bean Identity](image)

*Figure 5-23  Select RPG Procedures to externalize as Web service*

14. Click **Finish**. At this time the wizard generates a lot of code, so this operation can take a while.

**Note:** Alternatively, you can click **Next** if you want to see the rest of the Web service Wizards. In our example, we accept the default values for the rest of the wizard.
5.2.6 Testing the Web service Test Client

The wizard from the previous section generated a Web service and a JSP based Web service Test Client. Now we are ready to test the generated code:

1. After the wizard generates all the required artifacts, you see a Web Browser window. The wizard generates a Web service Test Client as a JSP based Web application. Click `reserveflight_XML(iseries.wsbeans.updateorder.RESERVEFLIGHTInput)` as shown in Figure 5-25.

**Note:** You can copy the URL into any Web browser, including Firefox or Internet Explorer®.
2. In the Inputs frame of the Web Browser window, enter the following values and click **Invoke** (see Figure 5-26):

- cUSTNUMBER: 500
- dEPARTDATE: 12/11/08
- aGENTNUMBER: 4
- dEPARTTIME: 7:12 AM
- fLIGHTNUMBER: 5191135
- tICKETS: 2
- sERVICECLASS: 2

**Note:** The values for `dEPARTDATE` and `dEPARTTIME` are specific to this sample. You **must** enter them as we note here. You can use Alternative Date mechanisms if you change the RPG program or Web service code to handle unique formats.

The Web service Client has demonstrated a successful Web service invocation because you receive the order number as the result.

**Note:** If you want to check whether the FLGHT400 Database was updated with the ReserveInfo, issue the following command on 5250 SQL session and scroll to the bottom of the file:

```
SELECT * FROM FLGHT400/ORDERS
```
5.2.7 Reviewing the generated Web service and Web service client code

At this point, WebSphere Development Studio client for iSeries has generated the Web service code, which in time might need some modifications. Thus, you might want to examine the code and begin to build a general understanding of the Web service implementation.

Reviewing the generated Web service code

Review the Web service and Web service generated client code using WebSphere Development Studio Client for iSeries:

1. Open Web Perspective by going to Window → Open Perspective → Web.
2. In Project Explorer expand Dynamic Web Projects → WebServiceProject → Java Resources → JavaSource → iseries.wsbeans.reserveflight. Spend some time reviewing the generated Web service Source files (see Figure 5-27).

3. The Web services Wizard generated various Java classes and two properties files:

   – **RESERVEINFO.java**: This class captures information that is stored in FLIGHTINFO data structure defined in the nfs001 RPG service program.
   – **RESERVEFLIGHTInput.java**: This class encapsulates input parameters to pass to the RPG service program.
   – **RESERVEFLIGHTResult.java**: This class encapsulates output parameters that are returned from the RPG service program.
   – **RESERVEFLIGHTServices.java**: This class performs RPG service program call using Toolbox classes.

Let us take a closer look at RESERVEFLIGHTServices.java.

Double-click **RESERVEFLIGHTServices.java**. Two methods in this Java Bean invoke nfs001 RPG program, the difference is in the method's return parameters:

   – reserveflight(...): returns RESERVEFLIGHTResult object
   – reserveflight_XML(...): returns a String object that includes an XML document, which is shown in Figure 5-28
The Web services Wizard generates two methods to give developers flexibility in using the Program Call bean. For example, in a Web services integration scenario, developers might want to use the method that returns an XML document. In addition, for the Java integration scenario, the method that returns a Java object is preferred.

Reviewing the generated JSP Web service client code

Now review the generated test code. This code helps you to test the Web service without writing the client’s part of the Web services invocation model. In addition, you can reuse the generated code in your own client’s application. To review the code, follow these steps:

1. In Project Navigator frame expand Dynamic Web Projects → WebServiceProjectClient → Java Resources → JavaSource → iseries.wsbeans.reserveflight. Spend some time reviewing the generated Web service Source files.

2. Expand Dynamic Web Projects → WebServiceProjectClient → WebContent → sampleRESERVEFLIGHTServicesProxy folder to look at the test clients JSPs.

**Important:** If you lose the URL for the Web service Test client, the following steps open the Web Browser with the correct URL:

1. Expand Dynamic Web Projects → WebServiceProjectClient → WebContent → sampleRESERVEFLIGHTServicesProxy.

2. Right-click TestClient and select Run → Run on Server.

3. Click Finish.
5.2.8 Deploying your Web service to WebSphere Application Server for i5/OS

When deploying the Web service, you need to update the WSDL for your production system. The wizard generates this file for testing in the WebSphere Test environment using `localhost` as the host name.

Modifying the WSDL document

*Note:* This step is not required to run the Web service. This step is required for Web service Clients to locate the Web service based on the WSDL document. The WSDL document should have the correct URI location specified.

To modify the WSDL document, follow these steps:

1. Make sure you are in Web Perspective.
3. Double-click `RESERVEFLIGHTServices.wsdl` to open in editor window. Graph view shows various elements of WSDL document in graphical format as shown in Figure 5-29.

![Figure 5-29 ReserveFlight WSDL](image-url)
4. In the Services section of the file, expand **RESERVEFLIGHTServicesService** → **RESERVEFLIGHTServices** and click the **wsdlsoap:address** element. The actual property and its value is shown and edited in the properties view as shown in Figure 5-30.

![Image of web service properties view](image)

**Figure 5-30  Change the URL destination**

5. Edit the location property and replace **localhost** with `<iSeries_Server>` and **9080** port with `<http_port>` (HTTP server port).

   **Note:** These values are unique to your system and the WebSphere Application Server profile that you have created on your server. See the WebSphere Application Server for i5/OS Information Center for additional information about profiles.

   If you are using the default profile created in i5/OS, the port is 9080; however, you must change the value for **localhost** to your system’s fully qualified host name.

6. Save the **RESERVEFLIGHTServices.wsdl** document by going to **File → Save** from the menu bar or by pressing CTRL+S on the keyboard.

7. Close **RERSERVEFLIGHTServices.wsdl**.

### 5.2.9 Modifying the Web service Client URI

Change Web service URI in Web service Client project. We need to change Web service URI, because the Web services wizard has generated a URI that points to the local host. This step is not required for Web services deployment, but it is a good practice to complete this step because it allows you to test the Web service with Web service client code that is
generated by the Web services wizard after the Web service has been deployed on a server. To modify the Web service Client URI, follow these steps:

1. In Project Explorer view expand **Dynamic Web Projects** → **WebServiceProjectClient** → **Java Resources** → **JavaSource** → **iseries.wsbeans.reserveflight**.
2. Double-click the **RESERVEFLIGHTServicesServiceLocator.java** file.
3. Modify **RESERVEFLIGHTServices_address** value (see Figure 5-31):
   - Replace **localhost** with your System i host name.
   - Replace **9080** port with the WebSphere profile port number (listen port for accessing WebSphere applications).

   ![Figure 5-31 Updating the location URI of the Web service](image)

4. Save **GETFLIGHTINFOServicesServiceLocator.java** by going to **File → Save** from the menu bar or by pressing CTRL+S on the keyboard.
5. Close **GETFLIGHTINFOServicesServiceLocator.java**.

### 5.2.10 Exporting the Web service EAR file

The Web Applications have been updated with correct values, so this section explains how to export the application so that you can install it on the production system. Typically, you export the Web applications in the form of an Enterprise Archive (EAR) file.
Follow these instructions to export the EAR file from WebSphere Development Studio client for iSeries:

1. In the Project Explorer view expand **Enterprise Applications** and right-click **WebServiceEAR**.
2. Select **Export → EAR file**.
3. On the EAR Export windows enter a destination: C:\temp\WebService.ear. Select the “Export source files” and “Overwrite existing file” options and click **Finish** as shown in Figure 5-32.

![Figure 5-32 Export the EAR file](image)

**Important:** By selecting the “Export source files” option, the entire application is encapsulated into a single EAR file. However, if you are going to redistribute this application to external customers, you should consider carefully whether to export the source.

4. You can export the Web services client EAR file in the same way. However, we use WebSphere Test environment in WebSphere Development Studio client for iSeries for testing our service.

### 5.2.11 Installing Web services application on System i platform

The application is located currently on the desktop of the client machine. Now, you will install it into i5/OS system using a Web Browser and the WebSphere Application Server administrative console.

**Note:** In our example, we use the *default* WebSphere profile. The administrative console for this profile runs on port 9060. If you use a different profile, use the administrative console port specific to your profile.
To install the Web service application into the WebSphere profile, follow these steps:

1. Open a browser, enter the administrative console URL:
   
   `http://<iSeries_Server>:<was_admin_port>/ibm/console`

2. In the Welcome panel, enter your user ID (if your profile is not secured, you can enter any string as your ID) and click Log in (see Figure 5-33).

![Figure 5-33 WebSphere Application Server Welcome window](image)
3. Expand **Applications** in the left navigation bar and click **Enterprise Applications**. Then, Click **Install** as shown in Figure 5-34.

![Figure 5-34 Install Enterprise Application](image1)

4. Select the **Local file system** radio button and specify C:\temp\WebService.ear or click **Browse** to locate the file. Then, click **Next** (see Figure 5-35).

![Figure 5-35 Ear file to install](image2)
5. On the “Choose to generate default bindings and mapping” panel click Next.
6. On the “Step 1: Select installation options” panel click Next.
7. On the “Step 2: Map modules to servers” panel:
   a. In the “Clusters and Servers” section, select both servers: WebSphere server and your HTTP server. (You can hold the Ctrl key and click both servers to select them.)
   b. Click Apply. This allows users to access your application through the external HTTP server.
   c. Click Next.
8. On the next panel click Step 4: Summary.
10. You should see a confirmation message similar to that shown in Figure 5-36 that the application was installed.

![Application WebServiceEAR installed successfully.
To start the application, first save changes to the master configuration.
Save to Master Configuration
To work with installed applications, click the 'Manage Applications' button.
Manage Applications](image)

Figure 5-36  Installation of Web service EAR file

11. Click Save to Master Configuration to save changes to the WebSphere Application Server configuration.
12. Confirm your choice by clicking Save on the next panel.
5.2.12 Starting your Web service application

You have successfully installed the Web application. However, by default the application is in the *Stopped* status. You need to start the application to access the Web service:

1. Expand **Applications** and click **Enterprise Application**.
2. Select **WebServiceEAR** and click **Start** (see Figure 5-37).

![Enterprise Applications](image)

*Figure 5-37  Start Web service application*

5.2.13 Testing the Web service on System i

The last step is to verify that the Web service works correctly. For this step we use WebSphere Development Studio client for iSeries:

1. Switch to the WebSphere Development Studio client for iSeries window.
2. Expand **Dynamic Web Projects** → **WebServiceProjectClient** → **WebContent** → **sampleRESERVEFLIGHTServicesProxy**.
3. Right-click **TestClient** and select **Run** → **Run on Server**.
4. Click **Finish**.
5. The WebSphere Test environment is started (if needed), and you should see the **TestClient.jsp** page in the browser window (see Figure 5-25 on page 57).
6. Click the **getEndpoint()** link.
7. Click **Invoke** (see Figure 5-38). This method returns the URL for your Web service. Verify that the returned value includes the correct host name and port number.

8. In the Method frame click **reserveflight_XML(iseries.wsbeans.updateorder.RESERVEFLIGHTInput)** as shown in Figure 5-39.
9. In the Inputs frame of the Web Browser window, enter the following values and click **Invoke** (see Figure 5-40):

- cUSTNUMBER: 500
- dEPARTDATE: 12/25/08
- aGENTNUMBER: 4
- dEPARTTIME: 7:12 AM
- fLIGHTNUMBER: 5191135
- tICKETS: 2
- sERVICECLASS: 1

![Figure 5-40   Input values](image)

The **Result** frame shows a confirmation number similar to Figure 5-40.

### 5.2.14 Adding additional Web services: GetFlightInfo and FindCustomers

In the previous sections, we demonstrated how to build the Web service for the flight reservation part of the application. For this service to work properly, you need to provide the departure time and flight number. These values cannot be received from a person calling to reserve a ticket. Thus, you need to add an additional Web service that allows you to search for this information based on customer data, such as the date, destination city, and preferred time of the day for departure.
To implement these services, you need to apply the same process (using the Web services wizard in WebSphere Development Studio client for iSeries) to other RPG procedures. Namely:

- **NFS404.RPGLE**: We use the FindFlightInfo or FindFlights procedures to get flight information.
- **NFS405.RPGLE**: We use the FindCustomers or GetCustNumber procedures for getting customer information.

FindFlightInfo, FindFlights, FindCustomers, or GetCustNumber RPG procedures are compiled into NFS400.SRVPGM.

To generate additional Web services, follow the instructions in 5.2.4, “Reviewing the RPG modules” on page 44, 5.2.5, “Creating and testing RPG Web service” on page 48, and 5.2.6, “Testing the Web service Test Client” on page 57.

**Important**: Make sure that you replace NFS001 with NFS400 in these instructions.

When you are done with this work, deploy the new Web services on the production server. Now your business partners can create an application that uses FindFlights, FindCustomer, and ReserveFlight procedures to look up and reserve a flight on behalf of their customer.

### 5.3 Exporting WSDL document (Optional)

The WSDL document includes several parameters that are required by the Web services clients to find and invoke this Web service over the Web. Thus, you should publish the WSDL document to the UDDI registry, e-mail it to your business partners, or place it on some HTTP server for other clients to access it. In this section, we show how to export this WSDL document from your project in WebSphere Development Studio client for iSeries.

In “Modifying the WSDL document” on page 61, we demonstrate how to modify the WSDL file before deploying your Web service application to the production server. After you have modified your WSDL file, follow the instructions here to export the modified WSDL file:

1. Start WebSphere Development Studio client for iSeries and open the Web perspective by selecting **Window → Open Perspective → Other → Web** and clicking OK.
2. In the Project Explorer view select `RERSERVEFLIGHTServices.wsdl` as shown in Figure 5-41.

![Figure 5-41  Select WSDL file](image)

3. On the WebSphere Development Studio client for iSeries menu, select **File → Export → File system** and then click **Next** (see Figure 5-42).

![Figure 5-42  File System Export](image)
4. Ensure that RESERVEFLIGHTServices.wsdl is selected in the next panel and click **Browse** (see Figure 5-43).

![Export File](image)

**Figure 5-43** Export File

5. Select **Desktop** and click **OK** as shown in Figure 5-44.

![Export to the desktop](image)

**Figure 5-44** Export to the desktop
6. Click **Finish** as shown in Figure 5-45.

![Figure 5-45 Export WSDL to Desktop](image)

You have now exported the WSDL document to the desktop, and you can forward it through e-mail or some other means to the various client developer teams for generating Web service Client to invoke the RPG Web service. The Web service Client that can consume the Web service include, but are not limited to, Java, RPG, C, C++, PHP, COBOL, and .NET.

### 5.4 Summary

In this Web service example, you extranlized an RPG procedure as a Web service. This Web service is accessible by any application that has access to your System i platform through port 9080 (default profile) and can communicate with a SOAP/HTTP protocol that adheres to the WS-I basic profile, including applications such as Java, J2EE Web Clients, JSF, JSP, .NET, and PHP.

In addition, you can build services to let your Web service clients explore even more business logic within your environment as we describe in 5.2.14, “Adding additional Web services: GetFlightInfo and FindCustomers” on page 70.
One of the simplest ways to build a Web service for your i5/OS applications is to use the DB2 Web services framework. This chapter list several scenarios for starting with DB2 Web services and illustrates some of these scenarios with the sample applications.
6.1 Reasons to use DB2 UDB based Web services

Why would you want to use DB2 UDB based Web services? There are certainly a few reasons and you might find even more. This section elaborates on just a few of the reasons why you would want to use DB2 UDB based Web services.

6.1.1 Get a feeling for the technology

This book, or any other book for that matter, could be the best technical publication ever written by human beings. Even if that were true, reading this book will not give you the same feeling for the technology as trying it out for yourself. A picture might be worth a thousand words, but some hands-on experience is worth even more!

“Playing around” with a small sample application can provide with a better feeling for the technology than pages of explanations and figures or graphics. Of course, this is not always possible, because it might be difficult for you to set up the environment. However, in this case, setting up the environment, at least if you have WebSphere Application Server and SQL installed on your system, should be fairly easy to do.

6.1.2 If you cannot modernize the whole application

By now you are probably aware of the fact that depending on the architecture of your current application or applications, you might have to envisage some modernization steps, such as converting your RPG OPM programs to the ILE architecture. Otherwise, it might be impossible to continue using them with the solutions that we introduce in this book, such as Web services based upon RPG programs.

However, this type of modernization might not be possible or might not even make sense in all cases. If the application in question is not functionally adequate, it probably does not make sense to modernize it. You would probably rather rewrite or re-engineer it.

If, nevertheless, you need to provide access to certain information in your database, information that continues to be maintained through your existing application, then you could create DB2 UDB based Web services to give access to that information without having to go through a potentially time and cost intensive modernization process.

6.1.3 There are strong SQL resources available

If in your team you have good SQL knowledge available, even developers without RPG knowledge at all, or if your RPG developers are not available because of other tasks, you might want to have your SQL-savvy person create SQL stored procedures and create Web services based on them. These Web services could then be used in other applications or to give external partners access to certain information through them.

6.1.4 You have invested in developing stored procedures

There are two types of the stored procedures in i5/OS: SQL and external (any system or service program or REXX™ procedure). Many companies develop host application using stored procedures. If your company has invested in stored procedures, there is a quick and easy way to externalize these stored procedures as Web services.

You can take advantage of DB2 Web services invoking stored procedures. In 6.4, “Creating a DB2 stored procedure” on page 95, we show an example of how to build an SQL store
procedure in WebSphere Development Studio client for iSeries and how to generate DB2 Web services to invoke this stored procedure.

6.2 Introducing the concepts and terminology

In this chapter, we use the terms that are typical in the context of DB2 UDB Web services discussion. To better understand the concepts described in this chapter, we define and explain these terms in this section.

6.2.1 DB2 Web services architecture overview

In the previous sections, we listed several business reasons for investigating and developing DB2 Web services solutions. There are also technical reasons for adopting this approach. IBM provides tooling and runtime support for DB2 Web services. Tooling and runtime support considerably simplifies the development and deployment effort for building DB2 Web services.

The runtime architecture of DB2 Web services is based on Web services Object Runtime Framework (WORF). WORF supports two paths for accessing DB2 UDB:

- Through DB2 XML Extender product
- Through a direct JDBC™ connection

Figure 6-1 shows the high-level architecture of WORF.

![WORF runtime architecture](image)

In the following sections, we explain all the components that are shown in Figure 6-1.
6.2.2 XML-based access and Document Access Definition (DAD)

This approach is based on the **DB2 XML Extender** functionality. DB2 XML Extender provides a mapping between elements in the XML documents and columns in the relational database. A special XML file, called Document Access Definition or DAD file, is used by DB2 XML Extender to specify:

- The operations that need to be performed
- The mapping between XML document structure and DB2 UDB table columns

The syntax of a DAD file is unique to DB2 XML Extender. The following XML-based operations are supported by DB2 XML Extender:

- **Query**: An XML-based query enables you to compose XML documents from relational data
- **Storage**: An XML document is broken down into its component parts and stored in relational tables

6.2.3 SQL-based access and Document Access Definition Extender (DADX)

A **Document Access Definition Extender**, or **DADX**, file is the extension of a DAD file. A DADX files specifies a Web service using a set of operations that are defined by SQL statements or DAD file. Effectively, a DADX file allows you to combine two distinct paths to access DB2 UDB for i5/OS in a single framework.

DADX approach is architected in the following way:

- First, you define a DADX group. This group is represented in a form of the directory. This directory includes several files, one of which is group.properties. This file defines the connection properties for the group. Example 6-1 shows a sample properties file. The most important properties are highlighted. As you can see, it includes the JDBC driver class, database URL, user ID, and password.

---

**Example 6-1 group.properties file**

```
#Tue Sep 19 11:31:36 CDT 2006
namespaceTable=namespacetable.nst
groupNamespaceUri=
reloadIntervalSeconds=5
dbDriver=com.ibm.as400.access.AS400JDBCDriver
dbURL=jdbc:as400:rchas10
enableXmlClob=true
useDocumentStyle=true
password=encoded:AwACAgQEBgYKCwgJDA0MDxQVEhMQFRTGBeExwVHhcwITIjNCUmJwgJCgsMDC4PcHFyczR1dnc4OT
o7PD0+Pw==
datasourceJNDI=initialContextFactory=
userID=test
autoReload=true
```
Second, you need to create one or more DADX files. All DADX files are placed in the group's directory. Each DADX file includes one or more operations. Remember that the operations can be SQL or XML based. Each DADX file is mapped to a single Web service. Example 6-2 shows the example of the DADX file with SQL based operation.

Example 6-2   Example of the DADX file

```xml
<?xml version="1.0" encoding="UTF-8"?>
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xsi:schemaLocation="http://schemas.ibm.com/db2/dxx/dadx dadx.xsd">
     <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
         <![CDATA[
            Query Customers
        ]]>
     </dadx:documentation>
     <dadx:operation name="QueryCustomer">
         <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
             <![CDATA[
            ]]>
     </dadx:documentation>
     <dadx:query>
         <dadx:SQL_query>
             <![CDATA[
                SELECT * FROM FLGHT400.CUSTOMERS WHERE FLGHT400.CUSTOMERS.CUSTOMER_NAME LIKE :CustName
             ]]>
         </dadx:SQL_query>
         <dadx:parameter name="CustName" type="xsd:string"/>
     </dadx:query>
 </dadx:operation>
</dadx:DADX>
```

The following SQL based operations are supported in the DADX file:

- call to a stored procedure
- insert
- update
- delete
- query

An SQL-based query allows you to send SQL statements, including stored procedure calls, to DB2 and to return the results with a default tagging. Data is returned using only a simple mapping of SQL data types, using column names as elements.

**Important:** SQL-based operations do not require DB2 XML Extender because there is no need for user-defined mapping of SQL data to XML elements and attributes.

### 6.2.4 Web services Object Runtime Framework (WORF)

The Web services Object Runtime Framework (WORF) enables you to define easily a basic Web service using standard SQL statements stored in an XML file to access local DB2 data. WORF provides an environment to create easily simple XML based Web services that access DB2 using the SOAP and the DADX.
WORF supports resource-based deployment of Web services, which means that you define your Web service in a resource file (DADX file) that you place in a directory of your Web application. When you request that resource file, WORF loads it and makes it available as a Web service.

If you edit the DADX file and request it again, WORF detects the change and loads the new version automatically. This process of automatically reloading the resource file makes Web service development more productive.

**Note:** With WORF you achieve dramatic decrease in the number of lines of code that you need to develop. Besides, you rely on well architected and tested framework. If there are any improvements in the framework, you get them without rewriting your applications.

### 6.2.5 Additional information about DB2 Web services

The following list provides links to additional information about DB2 Web services:

- **DB2 UDB for iSeries and Web services at:**
- **DB2 Web services for DB2 Practitioners (examples are given for DB2 UDB for Windows):**

### 6.3 Developing a DB2 UDB Web service

You can use the WebSphere Development Studio client for iSeries development tools, to develop a DB2 Web service. In this section, we show you how to create a DB2 UDB Web service based on a DADX file.

These are the steps to follow:

- Create a dynamic Web Project for the application
- Setup the DB2 connection
- Create an SQL Statement
- Configure the DADX Group
- Create the DADX file
- Generate the Web service based on the DADX file

We explain the steps in more detail on the following pages.

In 6.4, “Creating a DB2 stored procedure” on page 95, we provide a quick overview of how to create a DB2 stored procedure and then a Web service that uses the stored procedure.

### 6.3.1 Creating a dynamic Web Project for the application

First we need to create a dynamic Web Project in WebSphere Development Studio client for iSeries. Follow these steps:

1. Open WebSphere Development Studio client for iSeries.
2. Select **File** → **New** → **Project** from the menu.
3. In the pop-up window, select **Dynamic Web Project** and click **Next**.
4. Enter a name for the project, such as db2dadxws in our example in Figure 6-2 and click Finish.

![Dynamic Web Project window]

**Figure 6-2** Create and name a dynamic Web project

### 6.3.2 Setting up the DB2 connection

Next, you define a connection to the database that you use with your Web service:

1. In WebSphere Development Studio client for iSeries, switch to the J2EE perspective by clicking the window icon in the upper-right corner of the window and selecting **J2EE** (see Figure 6-3).

![Switch to the J2EE perspective]

**Figure 6-3** Switch to the J2EE perspective

**Note:** If you do not see J2EE in the list, select **Other**. A window opens. Select **J2EE** and click **OK**.

2. In Project Explorer, right-click the **Database Servers** folder and select **New Connection**.
3. Input a name for the connection you are creating, such as connDB2DADXWS, and click **Next**.
4. In the “Select a database manager” list box:
   a. Expand the **DB2 Universal Database™** subheading (Figure 6-4).
   b. Select **DB2 for iSeries V5R3**.
   c. In the “Host” field, input the host name of the database system.
   d. In the “Specify user information” group, enter the iSeries User ID and iSeries Password for the system that you entered in the previous step.
   e. Click **Test Connection** to verify that the setup is correct (see Figure 6-5). If the connection is successful, you should see a window. Click **OK**. Then, click **Next**.
5. In the “Include schemas that match the following conditions” group, click **Add**.

6. Enter FLGHT400% into the Filter text box as shown in Figure 6-6 and click **OK**.

![Add Schema filter](image)

**Figure 6-6  Add Schema filter**

7. Click **Finish**.

### 6.3.3 Importing the connection

When the DB2 connection setup is completed, you see your connection listed underneath the Database Servers folder in the Project Explorer. Next, you need to import it into your current project to make it available there:

1. Right-click the connection and select **Copy to project**.

2. Click **Browse** and select the name of the Dynamic Web Project. In our example, that is `db2dadxws` (see Figure 6-7).

![Copy to Project](image)

**Figure 6-7  Import the connection into the dynamic Web project**

3. Click **Finish**.
6.3.4 Creating an SQL statement

Next, you create the SQL statement that you want to use as the basis for your Web service. WebSphere Development Studio client for iSeries provides the wizard that guides you through SQL statement creation. In our example, we choose to manually enter a SQL statement. To create an SQL statement, follow these steps:

1. Select **File → New → Other**.
2. Expand the **Data** folder.
3. Select **SQL statement** and click **Next**.
4. For the drop-down box SQL statement, leave the default value SELECT. Select **Manually type an SQL statement**.
5. Deselect **Create new database connection**. Then click **Next** (see Figure 6-8).

![Create a new SQL statement](image1)

**Figure 6-8 Create a new SQL statement**

6. Select **Browse** in the Choose an Existing Database Model panel. In the Data resource selection panel, expand the folder named the same as the Dynamic Web Project (**db2dadxws** in our example) until you find the database that was created under the Databases folder in the Project Explorer (see Figure 6-9). Click **OK**.

![Select a database](image2)

**Figure 6-9 Select a database**
7. Click **Next**.

8. In the next panel, enter a name for your query. In our example, we enter **QueryCustomer**. Click **Next**.

9. Enter the following SQL statement:

   ```sql
   SELECT *
   FROM FLGHT400.CUSTOMERS
   WHERE FLGHT400.CUSTOMERS.CUSTOMER_NAME LIKE :CustName
   ```

10. When the statement is entered, click **Parse**.

11. Next, verify that the query is correct by clicking **Execute**. Click **Execute** again in the next window.

12. Double-click the first cell in the **Value** column and enter `Ba%` (see Figure 6-10). Click elsewhere in the table to deselect the current cell and click **Finish**.

   ![Figure 6-10 Specify the host variable value](image)

   *Figure 6-10 Specify the host variable value*

   The output of the query should look like that shown in Figure 6-11.

   ![Figure 6-11 SQL Query output](image)

   *Figure 6-11 SQL Query output*

13. After you have verified the output, click **Close**. Then click **Finish** to complete the SQL Wizard.

14. Close the SQL statement editor window by clicking **X** on the tab.
6.3.5 Configuring the DADX Group

This section explains how to configure the DADX group. Using the DADX approach, you save
time and effort in building DB2 Web services. IBM provides excellent development tools and
runtime support for this architecture.

The DADX group is a group that can include one or more DADX files. All files in the group
share the same set of the properties, namely: JDBC driver, DB2 system, and user ID and
password.

These are the steps to follow:

1. In WebSphere Development Studio client for iSeries, select File → New → Other.
2. Expand the Web services folder and select Web services DADX Group configuration.
   Click Next.
3. Select the folder named the same as the Dynamic Web Project, which is db2dadxws in our
   example. Click Add group.
4. Enter the name for the group. We use db2DADXgroup in our example (see Figure 6-12).
   Click OK.

![Figure 6-12 Name the DADX group](image)

5. Select the newly created group folder and click Group properties (see Figure 6-13).

![Figure 6-13 Select Group properties](image)
6. In the Group property window (Figure 6-14), replace the following:

- DB driver: com.ibm.as400.access.AS400JDBCDriver
- DB URL: jdbc:as400:<your host name>
  (replace <your host name> in this line with your system’s host name)
- User ID: enter your user ID
- Password: Click Edit. Enter the password for the user ID. Select the box labeled Store encoded and click OK.

![DADX Group Property](image)

**Figure 6-14   Set DADX Group properties**

7. Click **Finish**.

### 6.3.6 Creating the DADX file

Next, you create the DADX file that you use in the next section to generate the Web service:

1. In WebSphere Development Studio client for iSeries, select **File → New → Other**.
2. Expand the Web services folder and select **DADX File**.
3. Click **Next**.
4. In the Project frame select your project name, `db2dadxws` in our example (see Figure 6-15).

5. Enter a name for the DADX file and add a description.

6. Select **Generate a DADX file from the list of SQL queries or Stored Procedures**. Then click **Next**.

![Create DADX](image)

Figure 6-15  Enter a name and description for the DADX file
7. Expand the folder with the same name as the Dynamic Web Project (db2dadxws in our example) until you find the query that was created in the previous section. Select the query and click Next (see Figure 6-16).

![Figure 6-16   Selecting the SQL statement](image)

8. Click Next again and then click Finish.

9. The wizard now creates the DADX file and opens it in the editor’s view. The file should look similar to the code shown in Example 6-3.

Example 6-3   DADX file

```xml
<?xml version="1.0" encoding="UTF-8"?>
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xsi:schemaLocation="http://schemas.ibm.com/db2/dxx/dadx dadx.xsd">
    <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
        <![CDATA[
Query Customers
]]>
    </dadx:documentation>
    <dadx:operation name="QueryCustomer">
        <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
            <![CDATA[
            ]]
        </dadx:documentation>
        <dadx:query>
            <dadx:SQL_query>
                <![CDATA[
                SELECT * FROM FLGHT400.CUSTOMERS WHERE FLGHT400.CUSTOMERS.CUSTOMER_NAME LIKE :CustName
                ]]>}}"
10. Review the file and close it by clicking X on the editor's view tab.
11. If you want to open the file later, you can find it by expanding the Dynamic Web Projects folder in the Project Explorer as follows:
   - Expand your project in the Dynamic Web Projects folder. In our case it is db2dadxws.
   - Expand Java Resources \(\rightarrow\) JavaSource \(\rightarrow\) groups.db2DADXgroup.
   - There you find the file db2SQL.dadx.

### 6.3.7 Generating the Web service based on the DADX file

This section explains how to create a Web service based on the DADX file created in the previous section. Before generating the Web service, you need to specify the iSeries Web Tools Runtime configuration:

1. In the Project Explorer view expand Dynamic Web Projects folder.
2. Right-click the project name, which is *db2dadxws* in our example. Select Specify iSeries Web Tools Runtime Configuration.

![Figure 6-17 Specify iSeries Web Tools Runtime configuration](image-url)
3. In the next panel (see Figure 6-18), enter the host name, user ID, and password. Make sure the “Encode password encoding” option is selected. Then, click **Finish**.

![Configure Authentication](image)

*Figure 6-18  Set the host name, user ID, and password to configure the authentication*

4. In the Project Explorer view expand **db2dadxws → Java Resources → JavaSource → groups.db2DADXgroup**.

5. Right-click the **DB2sql.dadx** DADX file.

6. Select **New → Other**.
7. Expand **Web services** and select **Web service** (Figure 6-19). Then, click **Next**.

![Select a wizard](image)

Figure 6-19  Selecting Web service
8. Make sure the options are selected as shown in Figure 6-20 then click **Finish**.

**Attention:** You might see a warning message pertaining to the fact that the Web service does not comply with the WS-I SOAP Basic Profile. This is OK. Click **Ignore All**.

![Figure 6-20 Create the Web service](image)

The Web services wizard starts the WebSphere Test Environment and generates all artifacts for the Web service. This operation takes a while.
6.3.8 Testing the Web service

Next, test the Web service:

1. Expand the **Dynamic Web Project** folder (Figure 6-21).

![Figure 6-21 Location of the WSDL file](image)

2. Right-click the WSDL file and select **Web service → Test With Web services Explorer**.

3. In the Web services Explorer expand **theService → theSoapBinding** and click **QueryCustomer** (see Figure 6-22).

4. In the **CustName string** field enter Babc% and click **Go**. You should have three names returned.
6.4 Creating a DB2 stored procedure

This section explains how to first create a DB2 stored procedure and then how to create a Web service using this procedure. As you will see, this is pretty easy and most of the steps are almost identical to the previous example with an SQL statement.

**Note:** In this example, we assume that you have done the first example from this chapter that we describe in 6.3, “Developing a DB2 UDB Web service” on page 80. We reuse some of the objects that created from that example in the SQL statement example here.
6.4.1 Setting up the environment

Before you can begin developing stored procedures in WebSphere Development Studio client for iSeries, you must enable the database development capabilities. These capabilities are groups of functions that can be disabled or enabled as you need them. When you start the workbench for the first time, the database development capabilities are disabled.

Follow these steps to enable the database development capabilities:

1. Start WebSphere Development Studio client for iSeries.
2. From the menu bar select **Window → Preferences**.
3. Expand the **Workbench** node, and click **Capabilities**.
4. Expand the **Database Developer** node in the Capabilities list.
5. Select **Core Database Development** and **Stored Procedure and User-Defined Function Development** as shown in Figure 6-23.
6. Click **OK**.

![Figure 6-23   Enabling database capabilities](image)

6.4.2 Creating and building an SQL stored procedure

In the previous example, you created a dynamic Web project and a database connection. So you can skip these steps and move forward to the steps to create a stored procedure.

This section explains how to use a wizard to create a DB2 SQL stored procedure. This simple procedure receives two parameters and returns a result set.
Follow these steps to create the stored procedure:

1. If you followed the steps in the previous example, your workbench should display the J2EE perspective. Now, select **Window → Open Perspective → Data** to switch to the Data perspective.

2. In the Data Definition view navigate to the Stored Procedures folder: expand **db2dadxws → WebContent → db2dadxws → <connection name> → FLGHT400** (see Figure 6-24).

![Data Definition view](image)

Figure 6-24  Data Definition view

3. Right-click the Stored Procedure folder and select **New → SQL Stored Procedure**.

4. The new window displays. In the name field, enter a name for the stored procedure, such as `spSelectFlights` in our example. Select the “Build” and “Enable for use in DADX Web services” check boxes. Click **Next**.

![Creating a new SQL stored procedure](image)

Figure 6-25  Creating a new SQL stored procedure
5. On the New SQL stored procedure panel, select **One** in the Result set drop-down selection list. Then click **SQL Assist** as shown in Figure 6-26.

![Figure 6-26  Select Result set and SQL Assist](image)

6. In the Create A New SQL Statement panel, select the value **SELECT** in the SQL statement drop-down list and select the “Be guided through creating an SQL statement” option (see Figure 6-27). Click **Next**.

![Figure 6-27  Specify SQL statement information](image)
7. In the next panel, we compose the actual SQL statement. On the Tables tab, drill down in the FLGHT400 database until you can select the FLGHT400.FLIGHTS table. Click the > button to add it to the selected tables window (see Figure 6-28).
8. You do not need to select individual columns, and you do not need to create table joins. So, click the Conditions tab page and perform the following actions on this tab:

a. Enter these values on the first line (see Figure 6-29):
   i. In the Column field, double-click the first cell and select **FLIGHTS.DEPARTURE_INITIALS** from the drop-down list.
   ii. Double-click in the first cell of the Operator column. Select the equal sign (=) from the drop-down list.
   iii. Double-click the first cell in the Value column and enter :dept in the Value column.
   iv. Double-click the first cell in the And/Or column and select **AND**.

*Figure 6-29  Adding the conditions*
b. In the second line, enter the following values (see Figure 6-30):
   i. In the Column column, select `FLIGHTS.ARRIVALInicials`.
   ii. In the Operator column, select the equal sign (=).
   iii. In the Value column type `:arr`.

![Create A New SQL Statement](image)

Figure 6-30  Set the conditions for the SQL stored procedure

9. Click the Order tab.

10. Expand `FLIGHT400.FLIGHTS`, select `FLIGHT_NUMBER`, and click the > button. Now the results in the result set are ordered by flight number. Click **Next**.
11. The wizard displays the created SQL statement in an editor window where you could edit it (see Figure 6-31). You can click **Parse**, to parse the statement, but if you don’t edit it, this is not necessary. Click **Execute** to test the stored procedure. Click **Execute** again.

![Figure 6-31 Created SQL statement](image)

12. In the Specify Variable Values, enter the values **CHG** for the host variable :dept, and **ATL** for :arr (see Figure 6-32). Click in any other cell of the table. Then, click **Finish**. The result set is displayed in a separate window.

![Figure 6-32 Specify host variable values](image)
13. Click **Close** and then click **Finish**. The generated SQL statement is displayed in the New SQL Stored Procedure panel (Figure 6-33). Click **Next**.

![Figure 6-33 The created SQL statement](image)

14. The Parameters panel displays (see Figure 6-34). There is no need to change anything here. Click **Finish**.

![Figure 6-34 The specify parameters for the stored procedure panel](image)
When you are done with these steps, the SQL stored procedure is created and displayed in the editor’s view. The code should look similar to the one in Example 6-4.

**Example 6-4 SQL stored procedure code**

```sql
CREATE PROCEDURE FLGHT400.SPSelectFlights ( IN dept VARCHAR(16),
                                           IN arr VARCHAR(16) )

     RESULT SETS 1
     LANGUAGE SQL

-- SQL Stored Procedure
-- dept
-- arr

P1: BEGIN
   -- Declare cursor
   DECLARE cursor1 CURSOR FOR
       SELECT * FROM FLGHT400.FLIGHTS
       WHERE FLGHT400.FLIGHTS.DEPARTURE_INITIALS = dept
            AND FLGHT400.FLIGHTS.ARRIVAL_INITIALS = arr
            ORDER BY DEPARTURE ASC,
                    FLIGHT_NUMBER ASC;

   -- Cursor left open for client application
   OPEN cursor1;
END P1
```

### 6.4.3 Creating the DADX file and generate the Web service based on it

Now, that you have created the SQL stored procedure, you need to create the DADX file. This step is almost identical to the one described in 6.3.6, “Creating the DADX file” on page 87. Instead of repeating all these steps here again, we refer you to that section. Just remember to give the DADX file a different name than the one used in the previous sample.

When you have created the DADX file, you can follow the steps in 6.3.7, “Generating the Web service based on the DADX file” on page 90 to generate and test a Web service based on the DADX file.

### 6.5 Deploying the Web services

Before the Web services created in this chapter can be consumed, you must deploy them on the WebSphere Application Server. The whole process of deploying a Web service to WebSphere Application Server is described in detail in 5.2.8, “Deploying your Web service to WebSphere Application Server for i5/OS” on page 61. Therefore, we provide just an overview of the necessary steps in this section.
6.5.1 Modifying the WSDL file

The WSDL file is one of the main components of any Web service. It contains the URI of the Web service (its location on the Web), the methods that can be invoked, and so on. Any Web service client needs to read this file to learn how to use a Web service.

In our examples, we generated two WSDL files: `db2SQLService.wsdl` and `spSelectFlightsService.wsdl` (see Figure 6-35).

![Generated WSDL files](image)

To modify the WSDL file, follow these steps:

1. Double-click the `db2SQLService.wsdl` file. It opens in the WSDL Editor view.
2. Click the Source tab (at the bottom of the view). At the bottom of the source file, you should see the Service section, similar to Example 6-5.

   **Example 6-5  WSDL file segment**

   ```xml
   <service name="theService">
      <port binding="tns:theSoapBinding" name="theSoapPort">
         <soap:address location="http://localhost:9080/db2dadxws/db2DADXgroup/db2SQL.dadx/SOAP"/>
      </port>
      <port binding="tns:theGetBinding" name="theGetPort">
         <http:address location="http://localhost:9080/db2dadxws/db2DADXgroup/db2SQL.dadx/"/>
      </port>
      <port binding="tns:thePostBinding" name="thePostPort">
         <http:address location="http://localhost:9080/db2dadxws/db2DADXgroup/db2SQL.dadx/"/>
      </port>
   </service>
   
   3. Replace `localhost` with the host name of the system where you run WebSphere Application Server. Replace 9080 (the default port for the WebSphere Test Environment) with the port number of your WebSphere Application Server profile. Save the file.

4. Repeat this step for the second WSDL file.

Now your WSDL files reflect the correct information about the future location of your Web services. Any client that can connect to your WebSphere Application Server will be able to access these WSDL files, download them, and build the Web service client code to invoke these Web services.
6.5.2 Exporting the EAR file

Now when your application is ready for the production environment, you need to export it in a form of EAR file. Follow these steps:

1. Open WebSphere Development Studio client for iSeries.
2. Switch to the J2EE perspective.
3. Expand Enterprise Applications.
4. Right-click db2dadxwsEAR and select Export → EAR file (see Figure 6-36).

Figure 6-36   Selecting Export option

5. In the window that displays, specify the fully qualified path to the EAR file that will be created by the wizard. In our example, we enter c:\temp\db2dadxwsEAR.ear and click Finish (see Figure 6-37).

Figure 6-37   Exporting the EAR file

6. Open Windows Explorer and verify that the EAR file has been created in c:\temp.
6.5.3 Installing the application on WebSphere Application Server

To install the exported application on WebSphere Application Server profile, perform the following steps:

1. Start your WebSphere Application Server profile.
2. Open a browser and enter the WebSphere Application Server Admin Console URL, in the form:
   \[ \text{http://iSeries_Server:<was_admin_port>/ibm/console} \]
   In our example this would be:
   \[ \text{http://rchas10:9060/ibm/console} \]
3. On the administrative console login screen, enter your user ID (you can enter any ID you want) and click **Log in** (see Figure 6-38).

   ![Admin console login window](image)
   
   **Figure 6-38 Admin console login window**

4. On the WebSphere Administrative Console, expand **Applications** and click **Install New Application** in the navigation menu on the left hand side (see Figure 6-39).

   ![Administrative Console navigation menu](image)
   
   **Figure 6-39 Administrative Console navigation menu**
5. On the Preparing for the application installation panel, select **Local file system** and browse to the previously exported EAR file and select it and click **Open**. In our example, that location was `C:\Temp\db2dadxwsEAR.ear` (see Figure 6-40). Then, click **Next**. This step can take a little while, depending on network speed.

6. On the “Choose to generate default bindings and mappings” panel, click **Next**.

7. On the “Step 1: select installation options” panel, click **Step 4 Summary** (see Figure 6-41).
8. On the “Step 4: Summary” panel, click **Finish** (see Figure 6-42).

If the application is installed correctly, you should see a confirmation message (see Figure 6-43).

---

**Figure 6-42**  Summary of installation options

![Summary of installation options](image)

**Figure 6-43**  Installation confirmation

---

ADMA608I: The application db2dadxwsEAR is configured in the WebSphere Application Server repository.

ADMA503I: The library references for the installed optional package are created.

ADMA508I: The application db2dadxwsEAR is configured in the WebSphere Application Server repository.

ADMA509I: The application binaries are saved in /IBM/UserData/WebSphere/AppServer/VS/BB/ee/profile/default/temp/324f3207/workspace/cell/RCHAS10/applications/db2dadxwsEAR.ear/db2dad

ADMA505I: The application db2dadxwsEAR is configured in the WebSphere Application Server repository.

SECJ040I: Successfully updated the application db2dadxwsEAR with the appContextDForSecurity information.

ADMA501I: The cleanup of the temp directory for application db2dadxwsEAR is complete.

ADMA501I: Application db2dadxwsEAR installed successfully.

Application db2dadxwsEAR installed successfully.

To start the application, first save changes to the master configuration.

Save to Master Configuration

To work with installed applications, click the "Manage Applications" button.

Manage Applications
9. Click **Save to Master Configuration** to save the changes to the WebSphere Application Server configuration.

10. On the Save panel, click **Save** (see Figure 6-44).

![Figure 6-44   Save the configuration](image)

11. In the Administrative Console navigation, expand **Applications** and click **Enterprise Applications**. Locate the installed application (db2dadxwsEAR in our example), select the check box next to it, and click **Start** to start the application (see Figure 6-45).

![Figure 6-45   Start the application](image)

12. The next steps are additional (and not typical for other applications) steps for the DB2 Web services based on WORF and DADX. First, you need to create the classes folder under your profile’s directory in IFS. For our default server the directory is:

   `/QIBM/UserData/WebSphere/AppServer/V6/Base/profiles/default/classes`
13. Copy the following three JAR files that are bundled with your DB2 Web services applications to the new directory:
   - worf.jar
   - worf-servlet.jar
   - jt400.jar
   For example, copy the files from:
   /QIBM/UserData/WebSphere/AppServer/V6/Base/profiles/default/installedApps/
   <systemName>/db2dadxwsEAR.ear/db2dadxws.war/WEB-INF/lib/
   to
   /QIBM/UserData/WebSphere/AppServer/V6/Base/profiles/default/classes
   Now you should have three JAR files in your classes folder as shown in Figure 6-46.

14. Restart your WebSphere Application Server profile.

   **Note:** You need to copy three JAR files because of the way class loaders work in i5/OS. This step is not required if you install the DB2 Web services application to WebSphere Application Server on other platforms.
6.5.4 Testing Web services on the production server

After you deploy the Web services application on the production server, you can test them with Web services Explorer in WebSphere Development Studio client for iSeries:

1. Start WebSphere Development Studio client for iSeries and open the J2EE perspective.
2. Click the Launch Web service Explorer button on the toolbar (see Figure 6-47).

![Figure 6-47 Launching Web services Explorer](image1)

3. In the WSDL URL, enter the location of the WSDL file on your server. In our example running the application in the default profile the URL is:
   
   http://rchas10:9080/db2dadxws/wsd1/db2DADXgroup/db2SQLService.wsdl

4. Click Go (see Figure 6-48).

![Figure 6-48 Accessing the WSDL file](image2)
5. In the Navigator view of the Web services Explorer expand **theService** → **theSoapBinding** and click **QueryCustomer**.

6. Enter Babc% in the CustName field and click **Go** (see Figure 6-49).

You should see the results of your query in the Status frame of the view.

---

Figure 6-49  Testing Web service on the WebSphere Application Server profile
HATS Web service

This chapter describes a method to expose RPG application as a Web service using Host Access Transformation Services (HATS). It discusses the project considerations and investments for developing a Web service from traditional RPG application. Finally, the chapter demonstrates typical development steps in building such a Web service.
7.1 Project investments in developing a service

In wrappering existing business logic as a HATS Web service, you need to consider the following items for using SOA:

- Analyze the current state of the business application.
- Consider the time frame for the prototyping and production of a services application.

You need to consider also the application development, expense, and deployment for developing a Web service based application.

7.1.1 Analyzing an existing application

HATS Web service is a good solution where the existing applications can be driven using macros. With some applications, where it is not possible to drive the application using a macros, HATS Web service might not be a simple solution. You might require some modifications to the existing application flow to automate it using a macro.

Consider the following scenarios in our Flight Reservation System sample application:

1. Flight Reservation Inquiry

   This is a very simple scenario. The application window has only one input parameter and based on the input, it retrieves data from the database table. This application transaction can be driven easily with macro.

   Let us look at the transaction details and the navigation required in completing the transaction.

   a. Start personal communication session to your System i5.
   b. Run this command to add a library to the library list:
      ```
      ADDLIBLE  FLGHT400
      ```
   c. Call the Flight Reservation System application:
      ```
      GO  FLGHT400/FRSMAIN
      ```
d. Enter 3 - Inquire on an Existing Reservation on the Flight Reservation main menu panel and press Enter (see Figure 7-1).

```
FRSMAINX  16:27:57   Flight Reservation System   7/29/06   SYSTEMI5

Select one of the following:

1. Create a New Reservation
2. Update an existing Reservation
3. Inquire on an existing Reservation
4. Delete an existing Reservation
5. Fax Reservation Information

10. Flight Reservation System Maintenance

20. Reservation System Reports

90. Signoff

Selection or command

===> 3

Figure 7-1   Flight Reservation System main menu

```

e. You should see a logon panel. Enter Alex as agent name and mercury as the password. Press F10.

f. Enter the order number 005530217 and press F10 to work with selection (see Figure 7-2).

```
Figure 7-2   Order Selection panel

```
g. Review the order and press F3 or F12 to exit the Display Order panel (see Figure 7-3).

h. Press F3 to exit from Order Selection Panel.

<table>
<thead>
<tr>
<th>Flights Reservation System - Display Order</th>
<th>16:44:46  7/29/06 SYSTEMI5</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLIGHT INFORMATION</td>
<td>TICKET ORDER INFORMATION</td>
</tr>
<tr>
<td>Airline: DLT Flight: 5117009</td>
<td>Order Number.................: 005530217</td>
</tr>
<tr>
<td>Date of Flight..: 03 12 2004</td>
<td>Customer....: Aaronson, Linda</td>
</tr>
<tr>
<td>From City: Little Rock</td>
<td>Class of Service - First.....:</td>
</tr>
<tr>
<td>Depart Time.......: 07:54 AM</td>
<td>Business........:</td>
</tr>
<tr>
<td>To City...: Norfolk</td>
<td>Economy........: X</td>
</tr>
<tr>
<td>Arrival Time......: 09:54 AM</td>
<td>Number of Tickets............: 01</td>
</tr>
<tr>
<td>Number of Tickets..................: 01</td>
<td>Price $.....................: 169.00</td>
</tr>
<tr>
<td>Total Due w/ Tax $............: 175.76</td>
<td>Tax $.......................: 6.76</td>
</tr>
</tbody>
</table>

F3/F12=Exit

Figure 7-3 Display Order panel

2. Flight Selection Search and Flight Reservation Transaction

First let us look at the Flight Reservation Transaction and learn about the data entry fields and flow of the transaction:

a. Make sure that you are on Flight Reservation System main menu panel.

b. Enter 1 - Create a new Reservation on the Flight Reservation System main menu panel.

c. You should see a logon panel. Enter Alex as agent name and mercury as the password. Press F10.
d. Enter the date of the flight (Figure 7-4) for example:

09 12 2006

**Flights Reservation System - Create Order**  
17:01:58  7/29/06  SYSTEMI5

Type choices, press F10 to Make Reservation

<table>
<thead>
<tr>
<th>FLIGHT INFORMATION</th>
<th>TICKET ORDER INFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airline:</td>
<td>Flight: 0000000</td>
</tr>
<tr>
<td>Order Number........</td>
<td>PENDING</td>
</tr>
<tr>
<td>Date of Flight...:</td>
<td>09 12 2006</td>
</tr>
<tr>
<td>Customer...:</td>
<td></td>
</tr>
<tr>
<td>Class of Service -</td>
<td>First............:</td>
</tr>
<tr>
<td>From City.:</td>
<td>Business........:</td>
</tr>
<tr>
<td>Economy........:</td>
<td></td>
</tr>
<tr>
<td>Depart Time.......:</td>
<td>Number of Tickets.........</td>
</tr>
<tr>
<td>To City...:</td>
<td>Price $..................:</td>
</tr>
<tr>
<td></td>
<td>Tax $..................:</td>
</tr>
<tr>
<td>Arrival Time......:</td>
<td>Total Due w/ Tax $........:</td>
</tr>
</tbody>
</table>

F2=Refresh  F4=FROM Cities  F5=TO Cities  F6=Flights  F7=Customers

*Figure 7-4  Create order: Date of flight*
e. Press F4 to select From City. Enter 1 to select the city and press Enter (see Figure 7-5).

![Flights Reservation System - Create Order](image)

Type choices, press F10 to Make Reservation

**FLIGHT INFORMATION**

Airline: Flight: 0000000
Date of Flight...: 09 12 2006

From City.: Depart Time.......:
To City...:

Arrival Time......: F3=Exit

**TICKET ORDER INFORMATION**

**CITY SELECTION WINDOW**

Position To:
1=Select

<table>
<thead>
<tr>
<th>1</th>
<th>City Name</th>
<th>Initials</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Albany</td>
<td>ABY</td>
</tr>
<tr>
<td></td>
<td>Albuquerque</td>
<td>ALB</td>
</tr>
<tr>
<td></td>
<td>Anchorage</td>
<td>ANC</td>
</tr>
<tr>
<td></td>
<td>Atlanta</td>
<td>ATL</td>
</tr>
</tbody>
</table>

F2=Refresh F4=FROM Cities F5=TO Cities F6=Flights F7=Customers

*Figure 7-5  Create order: From City selection*
f. Press F5 to select To City. Enter 1 to select a city and press Enter (see Figure 7-6).

<table>
<thead>
<tr>
<th>Flights Reservation System - Create Order</th>
<th>17:09:46</th>
<th>7/29/06</th>
<th>SYSTEMI5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type choices, press F10 to Make Reservation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLIGHT INFORMATION</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Airline: Flight: 000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date of Flight.: 09 12 2006</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>From City.: Albany</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Depart Time.......:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>To City...:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arrival Time.......:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Position To:
1=Select

<table>
<thead>
<tr>
<th>City Name</th>
<th>Initials</th>
</tr>
</thead>
<tbody>
<tr>
<td>Albany</td>
<td>ABY</td>
</tr>
<tr>
<td>Albuquerque</td>
<td>ALB</td>
</tr>
<tr>
<td>Anchorage</td>
<td>ANC</td>
</tr>
<tr>
<td>Atlanta</td>
<td>ATL</td>
</tr>
</tbody>
</table>

More...

F2=Refresh F4=FROM Cities F5=TO Cities F6=Flights F7=Customers

Figure 7-6  Create order: To City selection
g. Press F6 to see all flights based on the date, from city, and to city that you entered earlier. Press Page Down to look at more flights. Enter 1 to select flight and press Enter (see Figure 7-7).

![Flights Reservation System - Create Order](image)

Figure 7-7   Create order: Flight selection
h. Press F7 to select Customer. Enter 1 to select a customer and press Enter (see Figure 7-8).

![Figure 7-8 Create order: Customer selection](image-url)
i. Enter class of service and the number of tickets (see Figure 7-9).

<table>
<thead>
<tr>
<th>Flights Reservation System - Create Order</th>
<th>17:19:01 7/29/06</th>
<th>SYSTEMI5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type choices, press F10 to Make Reservation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLIGHT INFORMATION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TICKET ORDER INFORMATION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Airline: AMA Flight: 2800143</td>
<td>Order Number.........: PENDING</td>
<td></td>
</tr>
<tr>
<td>Date of Flight...: 09 12 2006</td>
<td>Customer...: Aaronson, Lynn</td>
<td></td>
</tr>
<tr>
<td>From City.: Albany</td>
<td>Class of Service - First.........:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Business........:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Economy.........: X</td>
<td></td>
</tr>
<tr>
<td>Depart Time.......: 04:34 PM</td>
<td>Number of Tickets................: 01</td>
<td></td>
</tr>
<tr>
<td>To City....: Atlanta</td>
<td>Price $.....................: 269.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tax $.........................: 10.76</td>
<td></td>
</tr>
<tr>
<td>Arrival Time......: 06:34 PM</td>
<td>Total Due w/ Tax $............: 279.76</td>
<td></td>
</tr>
</tbody>
</table>

F2=Refresh  F4=FROM Cities  F5=TO Cities  F6=Flights  F7=Customers

Figure 7-9  Create order: Class of service and number of tickets
j. Press F10 to Make Reservation (see Figure 7-10).

k. Press F3 to exit the transaction.

Figure 7-10  Create order: Make reservation

Analyzing a transaction
We can divide the transaction that we describe in this section into two parts:

1. Flight selection
   Flight selection can be recorded as a macro which requires three input parameters:
   - Flight date
   - From city
   - To city
   These are input fields in the reservation transaction and allow you to add prompt action. Based on the three input parameters, the subfile displays for the flight selection. The rows in the subfile can be extracted as a table, because this part of application can be driven by macro and can be published as a Web service. We look at implementing this macro in detail in the Chapter 7.2, “Developing a HATS Web service” on page 126.

2. Complete Flight Reservation Transaction
   This transaction cannot be driven by a macro completely. As shown in this transaction, airline and flight numbers are not input fields, because they are display fields based on the flight subfile selection. Thus, display fields prompt action cannot be added. To enable prompt action, you must convert these fields to input fields, which requires some modifications to the existing flow of the transaction.
7.1.2 Naming conventions

When using HATS to create Web services, you need to follow these naming conventions:

- Macro names can begin with either an uppercase or a lowercase letter.
- Integration Object names become class names and must begin with an uppercase letter. Integration Object names are derived from macro names. So if the macro starts with a lowercase letter, then HATS converts the lowercase letter to uppercase when creating the Integration Object name.
- Macro prompt and extract names become method names and must begin with a lowercase letter.
- For any of the names mentioned here, a letter following an underscore (_) or a number must be in uppercase.

7.2 Developing a HATS Web service

You can use WebSphere Development Studio client for iSeries with HATS plug-in to develop a HATS Web service. This section explains how to create a HATS Web service based on a traditional RPG Applications.

**Note:** For installation requirements and instructions, see *IBM System i Application Modernization: Building a New Interface to Legacy Applications*, SG24-6671.

These are the steps to follow:

1. Create a HATS project
2. Record a HATS macros that will be used to:
   - Connect
   - Navigate through the host application
   - Extract data
   - Disconnect
3. Set Connection Properties
4. Create a HATS Integration Object
5. Create Web service support files
6. Create a Web service from the Integration Object.
7. Finally, test the Web service using the Web services Explorer

We will explain the steps in more detail in the following pages.

7.2.1 Creating a HATS project

The first step is to create a HATS project that includes all source code:

1. Open HATS Perspective.
   a. Click **Start** → **Programs** → **IBM WebSphere HATS 6.0** → **HATS Studio 6.0**.
   b. In the workspace launcher window, for workspace enter `c:\temp\redbook` and click **OK**.
   c. If you see a HATS Tip window, click **OK**.
   d. You should see HATS Perspective opened already. If you do not see it, select from the menu **Window** → **Open Perspective** → **Other** → **Host Access Transformation Services**.
2. Go through the New HATS project wizard:
   a. Launch the New HATS Project Wizard by clicking **File → New → Project**. Scroll down, expand **HATS** node, select **HATS Project** and click **Next** (Figure 7-11).

   ![Figure 7-11 New Project wizard](image)

   b. In Create a Project window in the Name field, enter **HATSWebService**.
   c. In the Target server field, select **WebSphere Application Server v6.0**.
d. Deselect the **Use default Enterprise Application project** check box and enter HATSWebServiceEAR as the name of the Enterprise Application (EAR) project name and click **Next** (see Figure 7-12).

![Create a Project: Project details](Figure 7-12) Create a project: Project details
e. In the connection settings window, enter the host name or IP address of the System i5 that is running the Flight Reservation application. For example, in the Host name field, enter systemi5.ibm.com. Select 5250 in the Type field (see Figure 7-13). Then, click Next.

Note: Starting with HATS 6.0.4, the product is repackaged as *WebFacing Deployment Tool with HATS Technology* (WDHT). The new product has HATS features but no OLTP capacity requirement for System i platform running i5/OS V5R4. In HATS studio the new connection type is shown as 5250W. To learn more about this product, see the following Web site:


![Figure 7-13  Create a project - Connection settings](image)

f. Because you are not planning to use this HATS application to perform screen transformation, accept the default template and click **Finish**.

g. Click **OK** in the HATS Tip window.
The HATS Project view shows all HATS projects in the workspace. Notice it now includes the project we just created (see Figure 7-14 on page 130).

![Figure 7-14  Project settings](image)

### 7.3 Recording macros

Next, you record a macro. A macro is a script that navigates through screens in an application. As you record the macro, you can identify which fields should be part of the input message of the eventual Web service and which fields should be part of the output message.

In general, it is recommended that you record three macros:

- A connect macro, which is responsible for logging onto the system and navigating to a start point
- A data macro, which performs the actual work of the transaction
- A disconnect macro which is responsible for signing off the system

Having separate macros allows for connection pooling. Pooling is a function in HATS that allows for a specified number of connections to be started and signed in. This specified number of connections allows for quicker transactions when a Web service request is made. Connection pooling is also more efficient because backend connections are reused, and socket connections are not being continually started to the backend system.
Recording a connect macro

Before recording your macro, use a terminal emulator, for example IBM Personal Communications (see Figure 7-15) or iSeries Access to sign on to the system using the same user profile as the one you intend to use with the macro. This forces the appearance of the Display Program Messages panel as you record your macro. Later in this section, we explain how to edit your macro to handle the case when the Display Program Messages panel does not appear.

![Figure 7-15 OS/400® Main Menu](image)

Select one of the following:

1. User tasks
2. Office tasks
3. General system tasks
4. Files, libraries, and folders
5. Programming
6. Communications
7. Define or change the system
8. Problem handling
9. Display a menu
10. Information Assistant options
11. iSeries Access tasks

90. Sign off

Selection or command

F3=Exit  F4=Prompt  F9=Retrieve  F12=Cancel  F13=Information Assistant
F23=Set initial menu
Perform the following instructions to record the macro:

1. From the HATS Project view, right-click your project's folder and select **Open Host Terminal** → main (see Figure 7-16).

   ![Figure 7-16 Open host terminal](image)

2. You might see a HATS Tip window. To keep this window from opening, select **Do not show any tips** and click **OK**.
3. Here you see the Sign On window from the host system. From here you start recording the macro. To start recording your Connect macro, click the **Record Macro** button on the toolbar (see Figure 7-17).

![Record Macro](image)

Figure 7-17   Record macro

4. On the Record Macro panel enter **SignOn** as the name. Click **Finish**.
5. For the first panel in the macro, HATS prompts you to supply the criteria to use for recognizing the screen. On the Define the starting screen panel, enter SignOn for the screen name (see Figure 7-18).

6. Select the criteria within a rectangular region (notice HATS has already selected an area in the upper, middle part of the screen). You can change this area if necessary. Click Finish.

7. On the Sign On panel, enter your i5/OS user name and password and press Enter.
8. Next, you see the Previous sign on information window. This window displays if the system value QDSPSGNINF (sign-on display information control) is set to 1. If you do not see this window, skip to step 12 on page 136.

From the toolbar of the host terminal window, click **Define screen Recognition Criteria** button (see Figure 7-19).

![Figure 7-19 Screen Recognition Criteria](image)
9. On the Select screen Recognition Criteria panel, enter SignOnInformation.

10. Notice this panel has a date and time on it. You should never include a date or time in a screen recognition if the date or time will change each time the screen is displayed. So, for this screen, drag your mouse around just the screen title, Sign-on Information, at the top. Select At a specified position and click Finish (see Figure 7-20).

![Figure 7-20 Define Screen Recognition Criteria]


12. Next, you see the Display Program Messages window. This window displays if there is already a session started using the same user name. If you do not see this window, skip to step 17 on page 137.

13. From the toolbar of the host terminal window, click Define screen Recognition Criteria.

14. On the Select screen Recognition Criteria panel, enter SignOnInformation.
15. Notice that this screen has a date and time on it. You should never include a date or time in a screen recognition if the date or time will change each time the screen is displayed. So, for this screen, you can either resize the existing dotted box by grabbing its corner with your left mouse button (the mouse pointer should change to double arrow), or you can drag your mouse around just the screen title, *Display Program Messages*, at the top. Select **At a specified position** and click **Finish** (see Figure 7-21).

![Figure 7-21 Define Screen Recognition Criteria](image1)

16. On the *Display Program Messages* panel, press Enter.

17. Next, you see the *i5/OS Main Menu* screen. Click **Stop Macro** to stop recording the macro (see Figure 7-22).

![Figure 7-22 Stop recording](image2)

18. You see the Define the exit screen of the macro window. Enter `i50SMainMenu`. 
19. Select the criteria within a rectangular region (notice HATS has already selected an area in the upper middle part of the screen). You can change this area if necessary. Click Finish (see Figure 7-23).

Figure 7-23 Define exit window
20. Click the **Save macro** button to save the macro (see Figure 7-24).

21. Next you test the macro that you just recorded. First, you need to sign off by entering 90 in the host terminal window. Then, execute the macro by pressing the **Play macro** button (Figure 7-24).

![Figure 7-24  Play macro](image)

You should see the macro is being executed, and finally you see the i5/OS Main Menu. You recorded the connect macro to connect to System i5. Next, you record a data macro.
Recording a data macro

In the data macro, you record the actual work of a transaction. This macro executes the application, defines the prompt actions for input fields, and defines the extract action for the output fields. To record a data macro, follow these steps:

1. Make sure that you are on i5/OS Main Menu in the host terminal window (see Figure 7-22).
2. To start recording your Data macro, click the Record Macro button on the toolbar.

![Figure 7-25 Record data macro](image)

4. On the Record Macro panel for Name enter FlightSearchData. Click Finish.
5. For the first screen in the macro, HATS prompts you to supply the criteria to use for recognizing the screen. On the Define the starting screen panel, enter i5OSMainMenu for the screen name.
6. Select the criteria within a rectangular region (notice that HATS has already selected an area in the upper, middle part of the screen). You can change this area if necessary. Click Finish.
7. On the i5/OS Main Menu Screen, enter following command:
   
   GO FLGHT400/FRSMAIN

8. Next, you see the Flight Reservation System main menu panel.
9. From the toolbar of the host terminal window, click the Define screen Recognition Criteria button.
10. On the Select screen Recognition Criteria panel, enter `FlightReservationSystemMainMenu` (see Figure 7-26).

11. Notice that this screen has no specific criteria selected. So for this screen, drag your mouse around just the screen title, *Flight Reservation System*, at the top. Select **At a specified position** and click **Finish**.

![Figure 7-26   Flight Reservation System main menu screen recognition criteria](image-url)
12. Type 1 on the command line to Create a new reservation and press Enter (see Figure 7-27).

![Figure 7-27 Create a reservation]

13. You see the Flights LOGON panel. From the toolbar of the host terminal window, click the Define screen Recognition Criteria button.
14. On the Select screen Recognition Criteria panel, enter FlightsLogon (see Figure 7-28).
15. Drag your mouse around just the screen title, Flights LOGON display, at the top. Select At a specified position and click Finish.

![Define Screen Recognition Criteria](image)

*Figure 7-28* Flights LOGON panel recognition criteria

16. In the Flight LOGON dialog box:
   a. Enter Alex as Agent Name.
   b. Enter mercury as the Password.
   c. Press F10 to log on.
17. On the Flight Reservation System - Create Order panel, click the **Define Screen Recognition Criteria** button and enter **CreateOrder** as the screen name. Use your mouse to select the text **Flights Reservation System - Create Order**. Click **Finish** (see Figure 7-29).

![Figure 7-29 Defining a recognition criteria for order window](image)

18. Up to this point, all input into the macro has been supplied by the developer. You now tell HATS that the Date of flight, from city and to city, will be supplied when the macro is run. You do this by adding a macro prompt. To add a prompt, click the **Add Prompt Action** from the Host Terminal toolbar (see Figure 7-31 on page 146).
19. In the Add Prompt Action dialog box, enter `flightDateMM` as the prompt name (eventually, this parameter is the input into the Web service) and click OK (see Figure 7-30).

**Note:** Make sure the prompt name starts with a *lowercase* letter.

![Figure 7-30 Define prompt action](image)

20. HATS prompts you to supply a value for the flight month, which allows you to continue recording the macro. Enter 09 into the field and click OK. You notice that this value is inserted into the month field and that the cursor moves to the day field.

21. Click the **Add Prompt Action** button from the Host Terminal toolbar.

22. In the Add Prompt Action dialog box, enter `flightDateDD` as the prompt name (eventually, this parameter is the input into the Web service) and click OK.

**Note:** Make sure the prompt name starts with a *lowercase* letter.

23. HATS prompts you to supply a value for the flight day, which allows you to continue recording the macro. Enter 12 into the field and click OK. You notice that this value is inserted into the day field and that the cursor moves to the year field.

24. Click the **Add Prompt Action** button from the Host Terminal toolbar.

25. In the Add Prompt Action dialog box, enter `flightDateYY` as the prompt name (eventually, this parameter is the input into the Web service) and click OK.

**Note:** Make sure the prompt name starts with a *lowercase* letter.

26. HATS prompts you to supply a value for the flight year, which allows you to continue recording the macro. Enter 2007 into the field and click OK. You notice that this value is inserted into the year field and that the cursor moves to the next field.
27. Click the **Add Prompt Action** button from the Host Terminal toolbar.

28. In the Add Prompt Action dialog box, enter `fromCity` as the prompt name (eventually, this parameter is the input into the Web service) and click **OK**.

   **Note:** Make sure the prompt name starts with a *lowercase* letter.

29. HATS prompts you to supply a city name, which allows you to continue recording the macro. Enter *Albany* into the field and click **OK**. You notice that this value is inserted into the From City field and that the cursor moves to the next field.

30. Click the **Add Prompt Action** button from the Host Terminal toolbar.

31. In the Add Prompt Action dialog box, enter `toCity` as the prompt name (eventually, this parameter is the input into the Web service) and click **OK**.

   **Note:** Make sure the prompt name starts with a *lowercase* letter.

32. HATS prompts you to supply a city name, which allows you to continue recording the macro. Enter *Atlanta* into the field and click **OK**. You notice that this value is inserted into the To City field (see Figure 7-31).

![Figure 7-31 Flight Search](image)

33. In Flights Reservation System - Create Order Screen, press **F6** to display flights.
34. On the Flight Reservation System - Flight Selection window panel, click the **Define Screen Recognition Criteria** button and enter *FlightSelectionWindow* as the screen name. Use your mouse to select the text *FLIGHT SELECTION WINDOW*. Click **Finish**.

35. On the toolbar click the **Record a loop** button (see Figure 7-32).

36. While recording your loop, pay very close attention to the instructions in the panel below the host screen.

37. You are already at the screen where the loop starts, so click **Next** at the bottom, right side of the window.

38. First, define the starting screen for the loop. Enter *FlightSelectionWindowLoop* as the screen name. Use your mouse to select the text *FLIGHT SELECTION WINDOW*. Click **Finish**.
39. Next, perform the actions that will be executed during each cycle of the loop. In the terminal window, drag your mouse around the data that you want to extract from the screen and click the Add Extract Action button (see Figure 7-33).

Figure 7-33  Selecting the region with data
40. On the Add Extract Action panel, type `flightSearchData` as the name and select **Extract this region as a table**. Click **Next** (see Figure 7-34).

![Add Extract Action - Settings](image)

**Figure 7-34  Add Extract Action - Settings**

41. You see the Table Extract Configuration window. In this window, you rename the column names to match the names on the host screen. Notice that some of the columns are extracted as two fields. You use this dialog box to merge those two fields into one field. Highlight each column name and change the name in the Column name field:
   a. Rename column1 to **ARL**.
   b. Rename column 2 to **Flight**.
   c. Rename column 3 to **Day**.
   d. Select Column4. Hold the Ctrl key and select Column5. Click the **Merge** button to merge these two fields into one (see Figure 7-35).
e. Now rename column 4 to *Departs*.

f. Select Column6. Hold the Ctrl key and select Column7. Click the **Merge** button to merge these two fields into one.

g. Now rename Column6 to *Arrives*. 

---

*Figure 7-35  Table Extract Configuration - Merge fields*
h. Rename Column7 to Price (see Figure 7-36)
   Also, notice that it allows you to expand or reduce the size of column to capture complete field from the screen.

i. Click Finish after you are done with all the changes.

![Figure 7-36 Table Extract Configuration](image)

j. Back at the terminal window, you need to add other actions that will be executed during each cycle of the loop. In this case, press the PgDnkey just once.

42. Click Next in the instructions panel (bottom, right side of the window).

43. For how to end the loop, select End when a unique screen is recognized and click Next.

44. Select Extract data from the last screen.

![Figure 7-37 Extract data from the last window](image)

45. Next, you must navigate to the last screen. Press the PgDnkey until you get to the last panel. In this example the last panel has the text Bottom at the bottom of the screen.

46. Click Next.

47. On the Define the final screen of the loop panel, enter FlightSelectionWindowBottom as the name.
48. Drag your mouse around the text *FLIGHT SELECTION WINDOW* and select **At a specified position** (see Figure 7-38).

49. Click **Next**.

![Figure 7-38 Flight Selection Window Bottom](image)

50. On the Recognition Criteria for Screen panel, click **Add** to define additional screen recognition criteria.

51. On the String Criterion panel, drag your mouse around the text *Bottom*. Select **At a specified position** and click **OK** (see Figure 7-39).

![Figure 7-39 Additional Screen Recognition Criteria](image)
52. On the Recognition Criteria for Screen panel, click **Finish**.

53. In the terminal window instruction panel, click **Finish** to complete the loop and continue recording the macro.

54. Now finish recording your macro by either pressing F3 or clicking the PF3 button.

55. Back in Flights Reservation System - Create Order panel, click the **Define Screen Recognition Criteria** button and enter `FlightsReservationSystemCreateOrder` as the screen name. Use your mouse to select the text *Flights Reservation System - Create Order*. Click **Finish**.

56. Press F3 to go back to Flight Reservation System main menu.

57. Back in Flights Reservation System - main menu panel, click the **Define Screen Recognition Criteria** button and enter `FlightReservationSystemMain` as the screen name. Use your mouse to select the text *Flights Reservation System*. Click **Finish**.

58. Press F3 to go back to i5/OS Main Menu.

59. Back in the i5/OS main menu screen, click the **Define Screen Recognition Criteria** button and enter `i5OSMain` as the screen name. Use your mouse to select the text *i5/OS Main Menu*. Click **Finish**.

60. Click the **Stop Macro** button to stop recording the macro.

61. Next, you see the Define the exit screen of the macro window. Enter `i5OSMain`.

62. Use your mouse to select the text *i5/OS Main Menu*. Click **Finish**.

63. Click the **Save macro** button to save the macro.

64. Next, test the macro that you just recorded. Make sure that you are at the i5/OS main menu screen and execute the macro by pressing the **Play macro** button.

65. You are prompted to supply prompt values. Enter following values (see Figure 7-40):
   a. `flightDateMM`: 09
   b. `flightDateDD`: 12
   c. `flightDateYY`: 2007
   d. `fromCity`: Albany
   e. `toCity`: Atlanta

66. Click **OK**.

![Figure 7-40  Supply Prompt Values](image)
You should see the results extracted in the Extract Result window (see Figure 7-41).

![Extract Results](image)

**Figure 7-41   Extract Results**

**Recording a disconnect macro**

The final step is to record a disconnect macro:

1. Make sure that you are on the i5/OS Main Menu before recording the disconnect macro.
2. To start recording your Disconnect macro, click the **Record Macro** button on the toolbar.
3. On the Record new macro or Insert window, select **New**.
4. On the Record Macro panel, for Name enter SignOff. Click **Finish**.
5. For the first screen in the macro, HATS prompts you to supply the criteria to use for recognizing the screen. On the Define the starting screen panel, enter i50SMain Menu for the screen name.
6. Select the criteria within a rectangular region (notice HATS has already selected an area in the upper, middle part of the screen). You can change this area if necessary. Click **Finish**.
7. On the i5/OS Main Menu panel, type 90 and press Enter.
8. Click the **Stop Macro** button to stop recording the macro.
9. Next, you see the Define the exit screen of the macro window. Enter SignOn.
10. Use your mouse to select the text Sign On. Click **Finish**.
11. Click the **Save macro** button to save the macro.
12. Next, you test the macro that you just recorded. First, sign on by selecting **SignOn** from the Play button drop-down menu (Figure 7-42).

![Figure 7-42 Execute sign on screen](image)

13. Execute the sign off macro by selecting **SignOff** from the Play button drop-down menu.

14. Close the Host terminal window.

### 7.4 Setting connection properties

When you deploy your Web service, it uses a connection (a 5250 session) to access your System i system. You set up the basic configuration parameters for your connection when you first built your project. Now you need to define connection pooling parameters and link the macros you just built to the connection.

#### 7.4.1 Enabling pooling

To improve performance for your Web service, it is recommended that you implement connection pooling for the connection that is used by the Web service. Connection pooling allows you to specify a number of connections (5250 sessions) that HATS maintains in a pool. Connection pooling avoids constant connecting and disconnecting from the host system in order to service multiple Web service requests.
The connections defined in your project are found in the **Connections** folder. HATS can support multiple connections for the purpose of collecting and combining data from multiple back end host sites. In this project you have defined just one connection, which by default is named **main**:

1. Double-click **main** under Connections (see Figure 7-43). Look through each of the tabs (at the bottom of the view) to see the different configuration settings. Click the Pooling tab.

![Figure 7-43  Connection Properties - pooling](image-url)
2. On the Pooling tab, click **Enable Pooling** and set the Connection Limits. For testing purpose set the limits to a **Minimum of 1** and a **Maximum of 2** (see Figure 7-44). For production use, you can set the limits to higher number based on the concurrent connection being used. Then click the Macros tab.

![Figure 7-44 Pooling - number of connections](image-url)
3. On the Macros tab (see Figure 7-45):
   a. For the Connect macro use the drop-down and select your Connect macro: SignOn
   b. For the Disconnect macro, select your Disconnect macro: SignOff.

![Figure 7-45  Connection Pooling - Macros](image)

4. Save main connection properties by selecting File → Save. Close the editor for your main connection by clicking the X on the main.hco editor tab.

7.5 Creating the integration object

A HATS Integration Object is a JavaBean that encapsulates a programmed interaction with a host application. Integration Objects can be used in multiple ways to integrate interaction with a host application into new Java or Web based programs. One use of an Integration Object is to provide the interaction with a host application for a Web service. HATS macros also provide a programmed interaction with a host application. In fact, creating a HATS macro is the first step in creating an Integration Object.

The Web service you are building in this example is intended to gather flights information from the host system based on flight date, from city and to city submitted by the Web service client. You have just finished creating the HATS FlightSearchData macro that does just that. Now all you need to do is tell HATS to create an Integration Object from your Data macro and then create a Web service from the Integration Object.
Follow these steps:

1. To create an Integration Object from your Data macro, in the HATS Project View expand the Macros folder, right-click `FlightSearchData` and select **Create Integration Object** (see Figure 7-46).

![Figure 7-46 Create Integration Object](image)

2. After HATS finishes creating the Integration Object, you find it in the **Source Integration → Object** folder. Notice it has the same name as the macro used to create it. Your Integration Object name is `FlightSearchData`. Due to the required naming conventions, if your macro had started with a lower case letter, HATS converts it to upper case in the Integration Object name.
7.6 Creating Web service support files

Before you create your Web service, you must first create HATS Web service support files:

1. In the HATS Project View, right-click your Integration Object **FlightSearchData** and select **Create Web service Support Files** (see Figure 7-47).

   ![Create Web service Support Files](image)

   **Figure 7-47** Create Web service Support Files

2. By default, the current Web project (**HATSWebService**) should be selected in the Project field. If it is not, select it. Enter **FlightSearchDataWS** into the Class name field. Click **Next**.
3. Place a check next to your Integration Object (FlightSearchData) to include it in the Web service class. Click **Finish** (see Figure 7-48).

*Figure 7-48  Select Integration Objects*
4. In the HATS Project View under *Source* notice the webserviceclasses folder. There are three classes created (see Figure 7-49):

- FlightSearchData_Input_Properties
- FlightSearchData_Output_Properties
- FlightSeachDataWS

![Figure 7-49 Web service Classes created](image-url)
7.7 Creating Web service

Now that you have created your Web service support files, you can create your Web service and deploy it to the WebSphere Test Environment. Follow these steps:

1. Click the Navigator tab (see Figure 7-49).

2. In the Navigator view, right-click your Web service class file (FlightSearchDataWS.java) in the Java Source\WebServiceClasses folder and select Web services → Create Web service (see Figure 7-50).

Figure 7-50   Create Web service
3. The first panel of the wizard displays basic settings for the Web service (Figure 7-51). At this point you, could click **Finish** to take all of the defaults, but select **Next** to view the different options that are available when creating a Web service.

![Figure 7-51 Web service Settings](image)

4. On the Object Selection Page panel, notice the bean name and click **Next**.
5. On the Service Deployment Configuration panel, notice that by default the server to which your Web service is deployed is WebSphere V6.0. Also notice your project name. Click Next (see Figure 7-52). This step can take some time; be patient.

Note: If the Web service runtime is not IBM WebSphere, click Edit and change the runtime.

6. On the Service Endpoint Interface Selection panel, click Next.

7. On the Web service Java Bean Identity panel, notice all of the Web service settings. The methods correspond to the Integration Objects you included in the Web service class file. Do not change any setting on this page and click Next.

8. At this point, your HATS application, including the Web service, is published to the WebSphere Test Environment, and the WebSphere server is started. This operation might take a few moments. After the test server is started, the Web service Publication panel appears. Click Finish.

HATS Web service is now deployed, and you can test it.

7.8 Testing the Web service

One output of creating your Web service is a WSDL file. You can find this file in the Navigator view in the Web Content\wsdl\webservice classes folder. You can use this WSDL file to test your Web service. Follow these steps:

1. Navigate to Web Content → wsdl → webservice classes folder under HATSWebService project.
2. Right-click the WDSL file, FlightSearchDataWS.wsdl, then select **Web services ➔ Test with Web services Explorer** (see Figure 7-53).

![Figure 7-53 Test with Web services Explorer](image)

**Note:** Alternatively, when you created the Web service, you could have selected the option **Test the Web service** in the first page of the wizard (see Figure 7-51 on page 164). This would have launched the Web services Explorer after creating and publishing the Web service.
3. The Web services Explorer displays in a Web browser. In the Actions area under Operations, click the `flightSearchDataProcessWS` link (see Figure 7-54).

![Figure 7-54 Invoking a Web service](image)

4. In the input field, enter the following values and click **Go** (see Figure 7-55):
   a. flightDateYY: 2007  
   b. flightDateDD: 12  
   c. flightDateMM: 09  
   d. fromCity: Albany  
   e. toCity: Atlanta

   **Note:** At this point a new connection is started to your backend system. Your Integration Object is instantiated and then navigates through the host application using the supplied data. If you notice that an error has occurred, then restart the server on the Server tab by right-clicking the server and selecting **Restart**.
Figure 7-55  Entering sample input data
5. Now you should see the result of the Web service in the Status pane of the window (see Figure 7-56).

![Invoking a WSDL Operation and Status Pane](image)

Figure 7-56 Web services output

**Note:** You can use the other tools within the Rational Software Development Platform to create a Java proxy, publish, or create a GUI front end for your Web service.

### 7.9 Next step

We demonstrated just one example of building a Web service. In real life, you need to create more than one Web service. Let us give you just one example using the Flight Reservation System application:

1. The Web service that we built returns the list of flights for certain date between two selected cities. However, your final goal is to be able to make a reservation. With HATS, your next step is to build another macro, *MakeReservation*. 
2. The MakeReservation macro starts with exactly the same sequence of steps as the FlightSearchData macro (see “Recording a data macro” on page 140). However, you continue with the Flight Reservation System application further (as described in 7.1.1, “Analyzing an existing application” on page 116) but adding prompts for other input fields, including:
   – Flight number (select from the list)
   – Customer name
   – Class of service
   – Number of tickets

   Your final step would be to submit a reservation and return a confirmation number.

3. Next, you create an IntegrationObject and Web service from your new macro, MakeReservation.

Now you have two Web services that you can combine in the client application:

1. Your client application displays a Web page with the same input fields that are required for the FlightSearchData macro: date of the flight (day, month, and year), to and from cities. When a user submit this page, your client application invokes the FlightSearchDataWS Web service.

2. The client application returns the list of available flights and let a user to select one.

3. After that your client application show a page with additional input field: customer name, class of service, and number of tickets.

4. When a user enters all data, you client application invokes the second Web service, MakeReservationWS, and pass input data for all fields:
   – Date
   – To city
   – From city
   – Flight number
   – Customer name
   – Class of service
   – Number of tickets

5. After executing the MakeReservationWS Web service, your client application returns a confirmation number.

By applying this technique, you can enable many host applications through HATS Web services.

7.10 Summary

In this Web service example, a 5250 application has been externalized as a Web service. As we saw in most of the cases, there will not be any modifications required to the 5250 applications. However, in some cases you might have to modify certain application flow so that it can be driven by a macro. This Web service is accessible by any application that has access to your System i server and can communicate with SOAP/HTTP adhering to the WS-I basic profile, including applications such as Java, J2EE Web Clients, JSF, JSP, .NET, PHP, and others.
PHP Web service

In this chapter, we describe how to create Web services using the scripting language PHP. Before diving into the samples, we provide a short overview of PHP to help you familiarize with it, should you not already know it.

In the second part of the chapter, we build Web services.
8.1 Introducing PHP

PHP is a scripting programming language that you can use to create Web sites. The PHP acronym stands for *PHP: Hypertext preprocessor*. The open-source language is used mainly for developing Web applications, and more recently, a broader range of modern software applications, such as the production and consumption of Web services.

PHP is very popular and used on over 22 million internet domains. Zend, the company behind the Zend Engine, estimates that there are approximately 2.5 million PHP developers in the world.

The founders of Zend, Zeev Suraski and Andi Gutmans, have been key contributors to the PHP language since 1997. PHP itself was invented in 1995 by Rasmus Lerdorf. Zend invests in the development of PHP itself, as well as in significant open source projects such as the Zend PHP framework and the Eclipse PHP plug-in. In addition, Zend delivers commercial products and services that enable developers and IT personnel to deliver and operate business-critical PHP applications.

8.2 Technology overview

In this section, we first have a look at how PHP works in a typical environment to provide a general idea about how the different components of a Web server work together. Then, based on the traditional “Hello, World” sample, we show you how PHP can be used. With these short samples, we hope to demonstrate one of the strengths of PHP—its *flexibility*.

8.2.1 How PHP works

Figure 8-1 on page 173 shows a simple diagram of how PHP works in a Web environment. Here are the steps in this process:

1. A site visitor requests a URL in his browser, and the request is transmitted over the Internet to the Web server.
2. The server parses the document. If there are PHP instructions (either embedded in HTML or in a pure PHP file), the code is transmitted to the PHP module. The PHP module processes the PHP functions.
3. In this sample, PHP accesses:
   - A database to read some data
   - Other i5/OS resource, such as data queue, host program, and so on
4. The data is returned from:
   - The database to PHP
   - Other i5/OS resource
5. PHP module returns it to the Web server as simple HTML output. The server embeds the result received in the document requested.
6. Finally, the requested document is sent back to the Web site visitor.
Of course, i5/OS access is not necessary in every case. Therefore steps 3 and 4 in Figure 8-1 might not be necessary.

### 8.2.2 What is needed to use PHP

There are not many requirements to start working with PHP as a Web development language. In fact, because PHP is a server-side scripting language, you only need a PHP-enabled server. PHP works with most Web server software, such as Apache or Microsoft's Internet Information Server. The source code of the PHP parser engine is the same on all operating systems, so there are no code changes required for a specific platform.

If you are interested in trying PHP for yourself on a smaller system than System i without spending too much time on installation and configuration tasks, you can search for the terms **LAMP** or **WAMP** using your favorite Internet search engine that are the abbreviations for Linux - Apache - mySQL - PHP or Windows - Apache - mySQL - PHP. Sometimes the *P* stands for **Perl** or **Python**, but these scripting languages are not the topic of this chapter. In a Macintosh environment, the terms can be **DAMP** or **MAMP** (Darwin or Macintosh, respectively). There are Web sites that offer combinations of these software packages for download. You might want to have a look at XAMPP ([http://www.apachefriends.org/en/xampp.html](http://www.apachefriends.org/en/xampp.html)), which is easy to install and quickly available to start working.

Alternatively, you can of course download the appropriate version of PHP from the site [http://www.php.net](http://www.php.net) and install it yourself. Because PHP is open source software, you can even download the source code and compile it yourself on the system of your choice.

### 8.2.3 There is more than one way to say “Hello, World”

Often, there is more than one way to do something, especially when it comes to programming. The sample of five programmers implementing five completely different programs to solve the very same problem has been cited often enough. PHP is no different. There probably is not the best or the worst solution. Depending on the context, one solution might be better than the other. Here we show some approaches to say “Hello, World.”

We assume that you have seen an HTML page before, and for the sake of clarity we only show the relevant code in our codes examples in this chapter, leaving off everything which is not necessary, such as DOCTYPEs. So the pages do not look very nice when displayed in a browser, but they should include everything for you to see the differences. Purists might argue that this is enough anyway for a Web page.
All of the following code samples should do nothing else than display a title and the text “Hello, World” underneath. The remarks might state the obvious, but it is our goal to show the various possibilities how PHP can be used in a Web-based environment. The code is not sophisticated at all, but all samples have been tested and should work as is.

**Just plain HTML**

Just to be complete, Example 8-1 shows a page using static HTML.

*Example 8-1  Static HTML*

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>IBM Redbook: Hello World</title>
</head>
<body>
<h3>Plain HTML</h3>
<p>Hello, World</p>
</body>
</html>
```

**HTML with embedded PHP**

Example 8-2 shows PHP code that is embedded in the HTML page. Copying and pasting the PHP code is a valid approach for smaller Web sites.

*Example 8-2  HTML with embedded PHP*

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>IBM Redbook: Hello World</title>
</head>
<body>
<h3>PHP embedded in HTML</h3>
<p><?php echo "Hello, World" ?></p>
</body>
</html>
```
HTML with embedded PHP, calling a self-written function

Example 8-3 shows an HTML document that includes embedded PHP. The PHP function is defined in an external PHP file (see Example 8-4 on page 175). Note that you need to reference the PHP file that includes the function using the `include_once` statement.

This approach allows you reuse the same function in many documents. If the function includes an error or needs to be modified, the change is done in one place and all documents using the function are immediately updated.

Example 8-3   HTML with embedded PHP function call

```php
<?php
    // Include the PHPFunctions file
    include_once("phpfunctions.php");
?>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>IBM Redbook: Hello World</title>
</head>
<body>
<h3>HTML calling an externally defined PHP function</h3>
<p>
<?php HelloWorld(); ?></p>
</body>
</html>
```

The file phpfunctions.php could of course include more than one function. In this example, we included a second function called `Hello10`, which displays the text `Hello, World` 10 times and then outputs the text `Done`. Note that the functions `printf` and `echo` are used to output the text.

Furthermore, the text `Hello, World` is followed by the HTML code for a break line `<br>`. This shows that you can perfectly use PHP to output HTML code that is interpreted correctly by the browser. In fact, you could create a very basic HTML page that is only calling one PHP function which then outputs massive amounts of HTML. The function is not used in the HTML file in Example 8-3, but you could easily exchange the name of the function to call it.

Example 8-4   The PHP function definition (file phpfunctions.php)

```php
<?php
function HelloWorld ()
{
    // Display "Hello, World"
    echo "Hello, World";
}

function Hello10 ()
{
    // Display "Hello, World" 10 times
    for ($i = 1; $i < 11; $i++)
    {
        printf("Hello, World<br>");
    }
    echo "Done.";
}
?>
```
PHP using HTML templates

The last example aims to show how to separate display from application logic by using PHP with an HTML template. The first step consists in creating a class `Template` with the appropriate functions (see Example 8-5). Our main method, `show($doc)`, receives the name of a document to be displayed as an argument. It sets the path, which is empty in our example, and then executes the template by using the PHP function `include()`.

After that it calls the function `showFooter()`, which prints a line with the current date and the current working directory.

```php
Example 8-5   Definition of class Template (TemplateClass.php)

```<php

class Template {

    public $template_dir;

    function show($doc) {
        $template = $this;
        include($this->template_dir.$doc);
        $this->showFooter();
    }

    function showFooter() {
        printf("<br><br><br>Today's date: %s | Current directory: %s", date('d. M Y'), getcwd());
    }
}
```

Example 8-6 defines the template file that we named `Template.tpl`. This file includes the display logic for the sample program.

```html
Example 8-6   HTML Template file

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title><?php echo $template->doctitle ?></title>
</head>
<body>
<?php echo $template->contentTitle; ?>
<?php echo $template->content; ?>
</body>
</html>
```
Now, all that is needed is the main document (see Example 8-7), which first includes the PHP template class file. Then, it creates a new instance of the class Template and assigns values to the class attributes. Finally, the show(‘Template.tpl’) function displays the template file (the display logic in the HTML template file), parsing and replacing the values according to what has been set in the created instance of TemplateClass.

Example 8-7   TemplateMain.php, our main file

```php
<?php
    include('TemplateClass.php');

    $template = new Template;
    $template->doctitle = 'PHP using an HTML template';
    $template->template_dir = '';
    $template->contentTitle = '<h3>PHP using an HTML template</h3>';
    $template->content = 'Hello, World';
    $template->show('Template.tpl');
?>
```

Using the techniques in Example 8-7, you get cleaner code and more agile applications, because display and application logic are separated. This separation allows you to change either the application logic or the look-and-feel of the application without affecting the other. As always, using templates might not be the best solution for everybody. As you can see with these simple examples, a template-based solution includes more code than one without templates. We do not use templates to write less code but to keep the display and application logic separate. These techniques follow Model-View-Controller (MVC) methodology. You can learn more about MVC at:

http://en.wikipedia.org/wiki/Model-view-controller

If you are interested in templates but do not want to write a solution yourself, then you are lucky. There are dozens of free template solutions available, Smarty being one of the better known template engines. Check out the following sites for template solutions:

http://framework.zend.com/

This site is the home of Smarty. The site includes interesting information, and we recommend reading the documentation that is available online:

http://smarty.php.net

Sitepoint is a publishing company in Australia that provides information for Web developers. The following Web site points to a thread in a discussion forum where site visitors have started building up a list with all PHP template engines:


At last count there were over 70 solutions in the list.

8.3 PHP on the System i platform

On 03 April 2006, IBM and Zend announced a multi-year agreement to deliver selected Zend PHP products and solutions for i5/OS. IBM and Zend have worked together to enable Zend Core and Zend Platform for i5/OS. For development, Zend Studio provides an integrated development environment (IDE) for building and debugging PHP applications. Zend Guard provides code protection and license management tools for independent software vendors (ISVs).
8.3.1 Zend Core for i5/OS

Zend Core is a fully tested and enhanced version of the open source PHP. It provides the PHP runtime and is packaged to make the software installation easier and faster with instant PHP setup.

The Zend Core for IBM has been enhanced to take advantage of i5/OS specific resources and capabilities and provides a seamless as it is shipped PHP development and runtime environment product supported by Zend and offering tight integration with DB2. The product includes native support for XML and Web services in support of increased adoption of service-oriented architectures (SOA).

More information about the Zend Core for i5/OS is available on the following Web site:
http://www.zend.com/products/zend_core/zend_for_i5_os

In addition to the software, you can also find information such as datasheets and white papers on the Zend Core for IBM. For developers it is worth having a look at the Developer Zone which features tutorials, documentation, discussion forums, and much more:
http://www.zend.com/developers.php

There is a specific forum for the i5/OS developers, Zend for i5/OS - Zend products for i5/OS, which is available at:

8.3.2 PHP version and availability

The Zend Core and Platform for i5/OS support PHP 5.0. Zend Studio for i5/OS is available today for i5/OS customers to start developing PHP applications. You can download Zend Studio for i5/OS at no cost from:
http://www.zend.com/products/zend_studio

Zend Core for i5/OS is available today at no cost. You can download it from the following Web site:
https://www.zend.com/core/oem_registration.php?access_code=IBMi5OS

For the i5/OS version, Zend provides three years of e-mail based support. Additional support options are available for purchase.

Required i5/OS release
Zend and IBM have delivered Zend Core and Zend Platform for i5/OS V5R3 and V5R4.

8.3.3 Accessing DB2 UDB and i5/OS resources

IBM and Zend are have worked together to deliver the i5/OS toolkit enabling PHP applications to access and use data easily in DB2 UDB for i5/OS. For more information about DB2 access read PHP: Zend for i5/OS, SG24-7327.

Furthermore, PHP applications can access easily and use:
- RPG and COBOL OPM and ILE applications in i5/OS
- Data Queue
- User Spaces
- Spooled File
- Active Jobs
This support is available in Zend Core for i5/OS.

8.3.4 Support for Zend products on i5/OS

Zend provides no cost e-mail based support for Zend Core for i5/OS and Zend Studio for i5/OS for three years. Zend also offers a variety of professional Support Programs for all of their products, ranging from software updates (delivery of major and minor updates, as well as bug and security hot fixes), Web-based support, business hours support, and premium 24x7 phone support.

8.4 The PHP Extension and Application Repository

The PHP Extension and Application Repository (PEAR) is a community-driven collection of open source classes that you can use to generate HTML code, make SOAP requests, create, validate and process HTML forms, send MIME mail, and much more. It was founded by Stig S. Bakken in 1999 and, since then, many people have joined the project.

8.4.1 Why PEAR important for you

PHP and the Zend Core for i5/OS provide many features, but you might find yourself in a situation where the needed functionality is not part of the core system. In general, that is the time when you start developing the missing features yourself.

Instead of doing so, you might first have a look at the PEAR Web site and browse the available categories. Every category includes one or more packages, each with a short description so that you can quickly browse through and eventually find the topic that you need. To give you just a small idea, here is a list of the top-level categories of the available PEAR modules:

- Authentication
- Date and Time
- Database
- Encryption
- Internationalization
- Logging
- Mail
- Networking
- Payment
- Web services
- XML

Again, even if this list already looks interesting, check out the PEAR Web site, because this is just an extract of the available categories. To directly access the list of available packages go to the following Web site:

http://pear.php.net/packages.php
8.4.2 Installing PEAR packages

One part of PEAR is a program that is called **pear** also is the PEAR package manager, which helps to download and install additional PEAR packages. i5/OS users can work with PEAR using the command line in PASE from /usr/local/Zend/Core/bin/pear.

8.4.3 Further information on PEAR

Further information on PEAR, its history, the available packages, and more is available at the following URL:

http://pear.php.net

If you do not know PEAR yet, we recommend that you read the About PEAR section at:


8.5 Creating a Web service with PHP

In this section, we discuss SOAP implementations in PHP, how to create PHP Web services, and what is available in Zend Studio for i5/OS to help facilitate Web services development. The first PHP Web service example is a simple service taking a string and integer as input and returning a string as output. The second example is more complex, wrapping an i5/OS program call and returning a complex data type.

8.5.1 PHP SOAP implementations

There is more than one SOAP implementation available for PHP 5. One is **NuSOAP**. NuSOAP is provided by NuSphere and Dietrich Ayala. It is a set of PHP classes. No PHP extensions are required that allow developers to create and consume Web services. We will not discuss NuSOAP here for two reasons:

- It is not part of PEAR
- It is not part of Zend Core for i5/OS

The SOAP implementation that we focus on is the PEAR SOAP Extension. PHP 5’s SOAP extension is the first attempt to implement the SOAP protocol for PHP in C. It has some advantages over the existing implementations written in PHP itself, the main one being speed. The extension is currently beta but should become more stable and reliable as time progresses. The SOAP extension implements a large subset of SOAP 1.1, SOAP 1.2, and WSDL 1.1 specifications. The key goal is to use the RPC feature of the SOAP protocol. WSDL is used where possible in order to make the implementation of Web Services more straightforward. Limited documentation is available at:


8.5.2 Zend Studio for i5/OS WSDL Generator

Zend Studio for i5/OS includes a WSDL Generator tool (see Figure 8-2). This tool is useful in the creation of WSDL files for a Web service, but it does have its limitations. The greatest limitation is that for the WSDL Generator to create the WSDL for a given PHP Web service, it needs more information than a PHP file normally includes because of the nature of PHP being a weakly-typed language. The additional needed information can be added to the code in the form of comments and additional classes with the appropriate comments.
Unfortunately, the WSDL Generator tool and the PHP syntax needed to support it are poorly documented. The PHP Web services examples in this chapter provide working examples of the format and syntax.

Figure 8-2 Zend Studio for i5/OS WSDL Generator is available from the Tools menu

Note: Use i5 toolkit templates (Web Service wrapper template) that are included in Zend Studio 5.5 for i5/OS to overcome limitations described in this section.

8.5.3 SOAP cache

Before developing a PHP Web service with the SOAP extension, it is important to disable the WSDL cache. If the SOAP WSDL cache is not disabled, the WSDL is cached for a period of time, even if the WSDL file is updated during development.

Go to the Zend Core Web Administration Console at (using default port number) and log in:

http://<host_name>:89/ZendCore
Click the Configuration tab and then the Extensions tab (see Figure 8-3). Find and expand the SOAP extension. Disabling the cache is shown in Figure 8-3 by setting `soap.wsdl_cache = 0` and `soap.wsdl_cache_enabled = off`. When development of the Web services is complete and the WSDL fill will no longer be altered, you need to re-enable the cache for better performance.

![Configuration for SOAP Extension in Zend Core for i5/OS console](image)

**Figure 8-3** Configuration for SOAP Extension in Zend Core for i5/OS console

### 8.5.4 Creating a simple Web service from PHP

We created this example using Zend Core Version 1.6.0, PHP Version 5.1.6. In this example, you create a simple PHP Web service using the SOAP extension in Zend Core for i5/OS that takes a string and an integer as input and returns a string. The Repeater Web service has two functions. The first repeats a string of text a number of times, and the second reverses a string and repeats it a number of times. Notice the syntax of the comment blocks in the `repeat` and `reverseRepeat` functions in Example 8-8. It is these comments in this specific format that the WSDL Generator needs to generate the WSDL for the Repeater Web service.

**Example 8-8  repeatServer.php Web service source code**

```php
<?php

class Repeater{

/**
 * Repeats a string of text a number of times
 * @param string $text
 * @param integer $times
 * @return string
 */

function repeat($text, $times){
    return str_repeat($text, $times);
}
```
function reverseRepeat($text,$times){
    return $this->repeat(strrev($text),$times);
}

//Expose as a web service
$server = new SoapServer("repeater.wsdl");
$server->setClass('Repeater');
$server->handle();
?>

To use file in the WSDL Generator Wizard, they must first be part of a Zend Studio for i5/OS project. You can add these files when creating a new project:

1. Select **Project → New Project** from the menu in Zend Studio for i5/OS (see Figure 8-4).

![Figure 8-4  Project menu in Zend Studio for i5/OS](image-url)
2. In the Editor window, enter the source code for the repeatServer.php file as shown in Example 8-8 on page 182.

3. Save this file on a server where you installed Zend Core for i5/OS. When you have written the Web service and added the files to a project, you can use the WSDL Generator in Zend Studio for i5/OS to create the associated WSDL file.

4. To launch the WSDL Generator Wizard select **Tools → WSDL Generator** from the Zend Core for i5/OS menu (see Figure 8-5).

---

**Figure 8-5**  
WSDL Generator on the Zend Studio for i5/OS Tools menu
5. In the first panel of the WSDL Generator Wizard, set the configuration name and the WSDL file name. For this example, we left the configuration name as the default and set the WSDL file name to `repeater.wsdl` (as shown in Figure 8-6), the same as in `repeatServer.php`. Click Next.

![WSDL Generator Wizard, setting the configuration name and the WSDL file name](image)

Figure 8-6  WSDL Generator Wizard, setting the configuration name and the WSDL file name

6. The next window in the WSDL Generator Wizard requires that you to point it to the PHP files that include the functions and classes that you want incorporated into the WSDL file (Figure 8-7). In this example, click the plus (+) button and select `repeaterServer.php`. Then, select the Repeater class.

7. Now, you need to add the URL Location. This is the URL where the PHP file is hosted by Zend Core for i5/OS. Then, click Finish.

![WSDL Generator Wizard, adding the files and classes to be exported to the WSDL](image)

Figure 8-7  WSDL Generator Wizard, adding the files and classes to be exported to the WSDL
The WSDL Generator Wizard saves the WSDL file that it creates to the local drive. In these examples, the WSDL file is referenced by a relative location, assuming that it is in the same directory as the Web service and the Web service client. So, you need to save the file to the same directory as the other PHP files on the System i server, /www/zendcore/htdocs by default. In subsequent runs of the WSDL Generator Wizard the existing WSDL file can be pointed at in the WSDL file name field, in which case the new WSDL is created in the correct directory.

When you click **Finish** (see Figure 8-7), the generated WSDL file is opened in the editor window. See listing in Example 8-9. Notice that the information included in the comments of repeaterServer.php is incorporated into the WSDL. You should not have to edit the WSDL file directly. If you change your PHP file, just rerun the WSDL Generator Wizard to update the WSDL file.

The generated WSDL file also points to the location of the Web service (highlighted). This is the URL that you typed in Figure 8-7.

**Example 8-9   The repeater.wsdl file**

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!-- WSDL file generated by Zend Studio. -->
<definitions name="repeater.wsdl" targetNamespace="urn:repeater.wsdl"
xmlns:typens="urn:repeater.wsdl" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="repeat">
    <part name="text" type="xsd:string"/>
    <part name="times" type="xsd:integer"/>
  </message>
  <message name="repeatResponse">
    <part name="repeatReturn" type="xsd:string"/>
  </message>
  <message name="reverseRepeat">
    <part name="text1" type="xsd:string"/>
    <part name="times1" type="xsd:integer"/>
  </message>
  <message name="reverseRepeatResponse">
    <part name="reverseRepeatReturn" type="xsd:string"/>
  </message>
  <portType name="RepeaterPortType">
    <operation name="repeat">
      <documentation>
        Repeats a string of text a number of times
      </documentation>
      <input message="typens:repeat"/>
      <output message="typens:repeatResponse"/>
    </operation>
    <operation name="reverseRepeat">
      <documentation>
        Reverses a string of text and repeats it a number of times
      </documentation>
      <input message="typens:reverseRepeat"/>
      <output message="typens:reverseRepeatResponse"/>
    </operation>
  </portType>
</definitions>
```
We explain the client code that uses the generated WSDL in 11.1.2, “Consuming the Repeater PHP Web service” on page 243.

8.5.5 Creating a PHP Web service to wrapper a program call

We created this example using Zend Core Version 1.6.0, PHP Version 5.1.6. In this example, you create a PHP Web service using the SOAP extension in Zend Core for i5/OS that takes three strings as input and returns a Complex Data Type. The ProgramCall Web service has one function. The function takes four parameters (first and last name, account number, and amount) as input, connects to the i5, uses i5_program_call to invoke INCRAMT (CL program). Example 8-10 shows the declaration part of the INCRAMT program.

Example 8-10 Declaration section of the INCRAMT program

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001.00</td>
<td>PGM</td>
</tr>
<tr>
<td>0002.00</td>
<td>DCL</td>
</tr>
<tr>
<td>0003.00</td>
<td>DCL</td>
</tr>
<tr>
<td>0004.00</td>
<td>DCL</td>
</tr>
<tr>
<td>0005.00</td>
<td>DCL</td>
</tr>
</tbody>
</table>
To return the results of INCRAMT, you need to use a complex data type, which is accomplished with the creation of the MyComplexDataType {} class (see listing in Example 8-11). Notice the syntax of the comment blocks above the variables in the MyComplexDataType class. Also notice that MyComplexDataType is referenced in the comments of the function callINCRAMT. It is these comments in this specific format that the WSDL Generator need to generate the WSDL for the ProgramCall Web service.

The most important parts of the PHP file are highlighted in Example 8-11.

Example 8-11   programCallServer.php source code for ProgramCall Web service

```php
<?php

class MyComplexDataType {
    /**
     * @var string
     */
    public $first;
    /**
     * @var string
     */
    public $last;
    /**
     * @var string
     */
    public $account;
    /**
     * @var string
     */
    public $ammount;

}

class ProgramCall{
    /**
     * Returns a complex data type
     *
     * @param string $firstNameInput
     * @param string $lastNameInput
     * @param string $ammountInput
     * @return MyComplexDataType
     */
    function callINCRAMT($firstNameInput,$lastNameInput,$ammountInput){
        
        $description = array(
            array("Name"=>"FIRST", "IO"=>I5_IN, "Type"=>I5_TYPE_CHAR, "Length"=>"15"),
            array("Name"=>"LAST", "IO"=>I5_IN, "Type"=>I5_TYPE_CHAR, "Length"=>"15"),
            array("Name"=>"ACCOUNT", "IO"=>I5_OUT, "Type"=>I5_TYPE_CHAR, "Length"=>"15"),
            array("Name"=>"AMOUNT", "IO"=>I5_INOUT, "Type"=>I5_TYPE_PACKED, "Length"=>"5.2")
        );
    }
```
$pgm = i5_program_prepare("I5SCHEMA/INCRAMT", $description);
if (!$pgm) die("<br>Program prepare error. Error number =".$i5_errno()." msg=".$i5_errormsg());

$parmIn = array(
  "FIRST"=>$firstNameInput,
  "LAST"=>$lastNameInput,
  "AMOUNT"=>$ammountInput
);
$parmOut = array(
  "FIRST"=>"FIRST",
  "LAST"=>"LAST",
  "ACCOUNT"=>"ACCOUNT",
  "AMOUNT"=>"AMMOUNT"
);

$ret = i5_program_call($pgm, $parmIn, $parmOut);
if (!$ret) die("<br>Program call error. Error number=".$i5_errno()." msg=".$i5_errormsg());

/* Close program call */
i5_program_close($pgm);

/* Close connection */
i5_close($conn);

**********************************************************
$results = new MyComplexDataType();
$results->first = $FIRST;
$results->last = $LAST;
$results->account = $ACCOUNT;
$results->ammount = $AMMOUNT;
return $results;
}

//Exposé as a web service
SERVER = new SoapServer("programCall.wsdl");
SERVER->setClass('ProgramCall');
SERVER->handle();
?>
After creating the programCallServer.php the WSDL Generator Wizard, you need to create the WSDL:

1. Select Tools → WSDL Generator. For this example we use programCall for the WSDL file name (see Figure 8-8). Click Next.
2. When programCallServer.php is selected, two classes are available. MyComplexDataType was created as an object to return complex data types, not a Web service in itself. So, you should not select it. You need to select ProgramCall and enter the URL Location with the location of the Zend Core for i5/OS host (see Figure 8-9). The URL should point to the location of this PHP file on your Web server. Click Finish.

![Image of WSDL Generator Wizard](image)

**Figure 8-9** WSDL Generator Wizard, adding the files and classes to be exported to the WSDL

The WSDL created in programCall.wsdl is worth looking at (see Example 8-12). Notice that MyComplexDataType is defined in the WSDL. This is possible because of several different things. First, the class MyComplexDataType was created, then it was commented in the right format for the WSDL Generator Wizard to parse, and finally MyComplexDataType is referenced as @return MyComplexDataType in function callINCRAMT (see Example 8-11 on page 188).

**Example 8-12** The programCall.wsdl file

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!-- WSDL file generated by Zend Studio. -->
<definitions name="programCall.wsdl" targetNamespace="urn:programCall.wsdl"
xmlns:typens="urn:programCall.wsdl" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsi="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:programCall.wsdl">
      <xsd:complexType name="MyComplexDataType">
        <xsd:all>
          <xsd:element name="account" type="xsd:string"/>
          <xsd:element name="ammount" type="xsd:string"/>
          <xsd:element name="first" type="xsd:string"/>
          <xsd:element name="last" type="xsd:string"/>
        </xsd:all>
      </xsd:complexType>
    </xsd:schema>
  </types>
</definitions>
```
Note: The Zend Studio 5.5.0 for i5/OS includes Web Service wrapper templates enabling calls to any i5/OS program.
Implementing Service Consumer

This part demonstrates several examples of how to create a Web service consumer. A service consumer invokes a Web service over the Web. A big advantage of Web services is platform independence, which allows you to build a service consumer using many different technologies.
IBM Web Services Client for ILE (RPG, C, C++, COBOL)

In i5/OS V5R3, IBM introduced IBM Web Services Client for ILE, which is part of XML Toolkit for iSeries. Web Services Client for ILE provides a set of libraries and Java tools that enable you to build C++ Web service client applications from existing Web Service Description Language (WSDL) files.

In this chapter, we demonstrate an example of creating an RPG program that invokes the ReserveFlight Program Call Web service using Web Services Client for ILE. Using Web Services Client for ILE is one way to enable an RPG program to participate in the SOA.
9.1 RPG as a Web service Client

The opportunity to invoke Web services from ILE applications (RPG or COBOL) has provided a consumable interface with IBM Web Services Client for ILE. In this chapter, we demonstrate an RPG program acting as a Web service client using Web Services Client for ILE.

The Web service that we demonstrate in this chapter is the ReserveFlight Web service, which you developed in Chapter 5, “ProgramCall (RPG, Cobol) Web service” on page 35. In this example one of the artifacts of the Web service is a WSDL document. This document is used to create a C++ stub to enable an RPG client code to invoke a Web service as shown in Figure 9-1.

![Figure 9-1 RPG as a Web service Client](image)

9.1.1 Development environment

There are several products and PTFs that you need to install prior to running this example. Ensure that the following products are installed and running at latest PTF level.

**Prerequisites**

This example assumes the following products and options with the latest group PTFs are already present in the environment:

- Qshell, 5722SS1 Option 30
- Host Servers, 5722SS1 Option 12
- IBM Developer Kit for Java, 5722JV1 Options 5 and 6
- IBM Toolbox for Java, 5722JC1
- 5722WDS Option 31 Compiler - ILE RPG IV
- 5722WDS Option 34 Compiler - RPG/400
- 5722WDS, option 52 can be used to compile C and C++ stubs
- 5722WDS, option 51 can be used to compile C only
- 5733XT1, option 9 XML Parser
- 5733XT1, option 12 XML Toolkit - Web services Client for C/C++

**Note:** PTF’s SI2445 and SI24421 for license program 5733XT1 must be installed to correctly generate the Web services C and C++ client.
In addition to the installed software, the Host Servers must be running on the target System i system.

The following products must also be installed on a development machine:

- WebSphere Development Studio Client for iSeries V6.0.1 (use Rational Product Updater to install latest fixes for V6.0.1)
- Firefox 1.0.7 or higher (or equivalent browser)

**Skills assumptions**

This technical reference assumes the following technical skills and knowledge:

- Experience with and basic knowledge of tools such as Rational Web Developer or WebSphere Development Studio Client
- Basic knowledge of i5/OS operating system administration
- Familiarity with the behavior and operation of the existing application that you want to modernize
- To deploy a Web service client you should possess the System i administrator skills.

**Deployment environment**

When you have developed an application, you need to have the following prerequisites in your deployment environment:

- You have installed and configured WebSphere Application Server - Express, Base, or Network Deployment (on any platform)
- System i platform on which the RPG application is deployed

### 9.2 ProgramCall bean example

In this example, we demonstrate Web service Client for C and C++ called from RPG and invoking the Web service *ReserveFlight* that you created in Chapter 5, “ProgramCall (RPG, Cobol) Web service” on page 35.

Follow these steps:

1. Go to **Start** → **All Programs** → **IBM Rational** → **IBM WebSphere Development Studio Client for iSeries V6.0** → **WebSphere Development Studio Client for iSeries**.
2. In the Workplace Launcher window in the Workspace field, enter `c:\temp\redbook` and click **OK** as shown in Figure 9-2.

![Figure 9-2 Start WebSphere Development Studio client for workspace](image)
9.2.1 Opening the J2EE Perspective

Prepare the WebSphere Development Studio client for iSeries workspace by opening the J2EE perspective:

1. Click Open perspective icon on the right hand side or Go to Window → Open Perspective and select Other as shown in Figure 9-3.

2. In the Select Perspective window, select J2EE and click OK.

9.2.2 Examining the WSDL document

**Important:**

When reviewing the WSDL document, you have two options:

- You can complete the instructions in 5.2.13, “Testing the Web service on System i” on page 68 and then you will have the same structure as described beginning with the steps that we list here.

- Alternatively, you can follow the instructions in 5.2.13, “Testing the Web service on System i” on page 68 but for your own application. In this case, the name and structure of the project will be different.

Review the WSDL document before exporting the document:

1. In Project explorer view, expand Dynamic web projects →WebServiceProject → WebContent → wsdl → iseries → wsbeans → reserveflight folder.

2. Double click RESERVEFLIGHTServices.wsdl to open in editor window. Graph view shows various elements of WSDL document in graphical format (see Figure 9-4).
Figure 9-4  ReserveFlight WSDL
3. The WSDL document can be edited in the source view or the graph view. Under services go to **RESERVEFLIGHTServicesService** → **RESERVEFLIGHTServices** and select the **wsdlsoap:address** element. The actual property and its value is shown in the properties view (see Figure 9-5).

![Figure 9-5 Web service URL](image)

4. If you follow the steps in 5.2.8, “Deploying your Web service to WebSphere Application Server for i5/OS” on page 61, the Web service location property should display URL for your Web service deployed on the server as it’s shown in our example in Figure 9-5.

   If you have not done the deployment yet, complete all steps in 5.2.8, “Deploying your Web service to WebSphere Application Server for i5/OS” on page 61 before continuing with the remainder of the instructions in this chapter.

### 9.2.3 Moving the WSDL file to the server

You need to move the WSDL document to System i platform to generate the C stubs and headers. You can copy the WSDL document, or you can open the Remote System Explorer perspective to move the document.
Copying the WSDL document
To copy the WSDL document, follow these steps:

1. In Project explorer view, expand Dynamic web projects → WebServiceProject → WebContent → wsdl → iseries → wsbeans → reserveflight folder.

2. Right-click RESERVEFLIGHTServices.wsdl and select Copy as shown in Figure 9-6.
Opening the Remote System Explorer perspective

To move the WSDL document using the Remote System Explorer perspective, follow these steps:

1. Click **Open perspective icon** on the right hand side or go to **Window → Open Perspective** and select **Other** as shown in Figure 9-7.

   ![Figure 9-7 Open Remote System Explorer Perspective](image)

2. In the Select Perspective window, select **Remote System Explorer** and click **OK**.


4. Right-click **Home** and select **New → Folder** shown in Figure 13-8.

   ![Figure 9-8 New Folder for WSDL file](image)
5. In the Folder Name field enter itso and click **Finish** as shown in Figure 9-9.

![Figure 9-9  Folder Name](image)

6. Right-click itso and click **Paste** as shown in Figure 9-10.

![Figure 9-10  Paste WSDL document](image)
9.2.4 Using Web Services Client for ILE to generate WSDL artifacts

The WSDL document is now located in the IFS directory /home/itso of System i server. The Web service Client for C and C++ needs to generate the stubs and headers. These files are used by an RPG program to consume a Web service:

1. Open 5250 client session to System i platform.
2. At the command prompt, enter QSH.
3. Run the cd /home/itso command.
4. Run the following command:
   `/QIBM/ProdData/xmltoolkit/WSCI-1.0-OS400/bin/wsdl2ws.sh RESERVEFLIGHTServices.wsdl -lc`
5. You should see some output in the terminal window. At the end, you should see the following message:
   'Code generation completed.'
6. Run the ls command. You should see generated files.

Note: You can examine the generated code more closely by refreshing the Remote System Explorer view in WebSphere Development Studio client for iSeries, File → Refresh.

9.2.5 Creating an RPG program to invoke a Web service

Using WebSphere Development Studio client for iSeries, you can now generate an RPG program that consumes the RESERVEFLIGHTServices Web service. The RPG program makes a single reservation:

1. Click Open perspective icon on the right hand side or go to Window → Open Perspective and select Other as shown in Figure 9-11.

   ![Figure 9-11 Open Remote System Explorer Perspective](image)

2. In the Select Perspective window, select Remote System Explorer and click OK.
4. Right-click iSeries Objects and select New → Library (see Figure 9-12).

![Figure 9-12  New Library](image)

5. On the iSeries Library wizards enter the following:
   a. In Library field, enter RPGTEST.
   b. Select Test.
   c. In Text field, enter RPG Web service Client Test.
   d. Click Finish as shown in Figure 9-13.

![Figure 9-13  Create a New Library](image)
6. Right-click **RPGTEST.*lib.test** and select **New → Source Physical File** as shown in Figure 9-14.

![Figure 9-14  New Source Physical File...](image)

7. On iSeries Source Physical File wizard enter the following values and click **Finish** as shown in Figure 9-15:
   - File: QRPGLESRC
   - Record Length: 500

![Figure 9-15  New source file](image)
8. Right-click QRPGLESRC.*file pf src and select New → Member as shown in Figure 9-16.

![Create New Member](image)

Figure 9-16  Create New Member

9. In the iSeries Source Member enter the following values and click Finish (see Figure 9-17):

- Member: RPGTEST
- Member Type: RPGLE
- Text: RPG Web service Client Test

![Create RPG Source Member](image)

Figure 9-17  Create RPG Source Member
10. RPGTEST.RPGLE opens in the Editor's window (see Figure 9-18).

12. Double-click RESERVEFLIGHTServices.h.
13. Examine the generate code and note C externalized functions (see Figure 9-19):

- Functions relating to Web service client proxy
- Functions relating to Web service methods

These functions will be used by an RPG program to invoke the Web service.

Figure 9-19  Generated C code for RERSERVEFLIGHTServices.h

14. Click the RPGTEST.RPGLE tab and copy the source code from Example 9-1. Note how the C stubs are externalized and used throughout the RPG code.

Example 9-1  RPG Source code to invoke Web Services Client for ILE

```rpg
H DFTNAME(GETCINFORP)
/INCLUDE RPGINCLUDE
*******************************************
D RFLInput   DS
*******************************************
D AgentNumber1                *
D CustNumber1                  *
D FlightNumber1                 *
D DepartDate1                  *
D DepartTime1                  *
D Tickets1                      *
D ServiceClass1                 *
D
*
D AgentNumber                  9B 0
D CustNumber                   9B 0
D FlightNumber                 8
D DepartDate                   9
D DepartTime                   9
D Tickets                      4F
D ServiceClass                  2
*
D getStub               PR        ExtProc('get_RESERVEFLIGHTServices_+

```
D pEndpoint  *  Value  
  
D getRFlight  PR  *  ExtProc('reserveflight_XML')  
D pRFLWS     *  Value  
D pRFLInput  *  Value  
  
D destroyStub  PR  ExtProc('destroy_RESERVEFLIGHTSerti+ 
    ces_stub')  
D pCityInfoWS *  Value  
  
D destroyParm  PR  ExtProc('axiscAxisDelete')  
D value       *  Value  
D description 9B 0 Value  
  
D RFLWS       S  *  
D RFLInput1   S  *  
D Endpoint    S  100A  
D pOrderXML   S  *  
D OrderXML    S  120A  
D Output      S  52A  
  
C  eval  AgentNumber = 5  
C  eval  CustNumber = 500  
C  eval  FlightNumber = '5191135' + X'00'  
C  eval  DepartDate = '12/11/08' + X'00'  
  
C  eval  DepartTime = '7:12 AM' + X'00'  
C  eval  Tickets = 3  
C  
C  eval  ServiceClass = '1' + X'00'  
  
C  eval  AgentNumber1 = %addr(AgentNumber)  
C  eval  CustNumber1 = %addr(CustNumber)  
C  eval  FlightNumber1 = %addr(FlightNumber)  
C  eval  DepartDate1 = %addr(DepartDate)  
C  eval  DepartTime1 = %addr(DepartTime)  
C  eval  Tickets1 = %addr(Tickets)  
C  eval  ServiceClass1 = %addr(ServiceClass)  
  
C  eval  Endpoint = 'http://itso:9080' +  
        '/WebServiceProject/services'+  
        '/RESERVEFLIGHTServices' + X'00'  
C  eval  RFLWS = getStub(%Addr(Endpoint))  
C  eval  RFLInput1 = %Addr(RFLInput)  
C  eval  pOrderXML = getRFlight(  
    RFLWS :  
    %addr(RFLInput1))  
C  eval  OrderXML = %str(pOrderXML)  
C  if (OrderXML <> *blanks)  
    movel *blanks field 52  
C  eval  Output = 'The order information: '  
  
C eval field = %subst(OrderXML:61:7)
C
C Output dsply
C field dsply
C else
C eval Output = 'There is no order information '
C Output dsply
C endif
C callp axiscAxisDelete(pOrderXML:XSDC_STRING)
C eval pOrderXML=*NULL
C callp destroyStub(RFLWS)
C seton lr

Important: You need to adjust the value of itso:9080 for the correct values for your System i platform and WebSphere Application Server profile values.

Web Services Client for ILE INCLUDE file for RPG
The RPGINCLUDE file has been created to import some RPG constants and RPG functions that is common to running Web service Client in RPG Service Programs (see Example 9-2).

Note: In our example source code, we used the RPGINCLUDE file for axiscAxisDelete(pOrderXML:XSDC_STRING) function. The function deletes the return values after they have been processed so as not to create a memory leak for the RPG program.

Review the WSCI-1.0.pdf for memory considerations. The file is located at /QIBM/ProdData/xmltoolkit/WSCI-1.0-OS400/docs/WSCI-1.0.pdf and can be access using a mapped drive or FTP.

Example 9-2 RPGINCLUDE

* LICENSE AND DISCLAIMER
  * ----------------------
  * This material contains IBM copyrighted sample programming
  * source code ( Sample Code ).
  * IBM grants you a nonexclusive license to compile, link,
  * execute, display, reproduce, distribute and prepare derivative
  * works of this Sample Code. The Sample Code has not been
  * thoroughly tested under all conditions. IBM, therefore, does
  * not guarantee or imply its reliability, serviceability, or
  * function. IBM provides no program services for the Sample Code.
  *
  * All Sample Code contained herein is provided to you "AS IS"
  * without any warranties of any kind. THE IMPLIED WARRANTIES OF
  * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
  * NON-INFRINGEMENT ARE EXPRESSLY DISCLAIMED.
  * SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED
  * WARRANTIES, SO THE ABOVE EXCLUSIONS MAY NOT APPLY TO YOU. IN
  * NO EVENT WILL IBM BE LIABLE TO ANY PARTY FOR ANY DIRECT,
  * INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES FOR ANY USE OF
  * THE SAMPLE CODE INCLUDING, WITHOUT LIMITATION, ANY LOST
  * PROFITS, BUSINESS INTERRUPTION, LOSS OF PROGRAMS OR OTHER DATA
  * ON YOUR INFORMATION HANDLING SYSTEM OR OTHERWISE, EVEN IF WE
  * ARE EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
D* Delete datatype storage, required to free memory resources
D* returned by deserializer.
D* ---------------------------------------------------------------
D axiscAxisDelete...
D  PR          10I 0  EXTPROC('axiscAxisDelete')
D* Returns return code 0=succes
D  XSDPTR                        *  VALUE
D* xsd type to delete
D  XSDTYPE                     10I 0  VALUE
D* xsd type

D* Allocate storage for datatype.
D* ---------------------------------------------------------------
D axiscAxisNew...
D  PR                        *  EXTPROC('axiscAxisNew')
D* Returns pointer to storage
D XSDTYPE  10I 0 VALUE
D* Object type
D LENGTH   10I 0 VALUE
D* Length (0 for non-char buffers)

D* Register exception handler routine.
D* ---------------------------------------------------------------
D axiscAxisRegisterExceptionHandler...
D                 PR                   EXTPROC('axiscAxisRegisterException+
D                                     Handler')
D  FUNCPTR                *    VALUE
D* Function pointer

* Prototypes and definitions from axis/GDefine.h (partial)
* Prototypes and definitions from axis/TypeMapping.h (partial)

D* Error codes.
D* ---------------------------------------------------------------
D AXISC_SUCCESS...
D C                   0
D* AXISC_FAIL...
D* C                  -1
D AXISC_OBJECT_ALREADY_EXISTS...
D C                   1
D AXISC_NO_SUCH_HANDLER...
D C                   2
D AXISC_NO_SUCH_SERVICE...
D C                   3
D AXISC_NO_REMAINING_SOAP_HEADERS...
D C                   4

D* XSDC_UNKNOWN...
D C                   1
D XSDC_INT...
D C                   2
D XSDC_FLOAT...
D C                   3
D XSDC_STRING...
D C                   4
D XSDC_LONG...
D C                   5
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Character</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSDC_SHORT</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>XSDC_BYTE</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>XSDC_UNSIGNEDLONG</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>XSDC_BOOLEAN</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>XSDC_UNSIGNEDINT</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>XSDC_UNSIGNEDSHORT</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>XSDC_UNSIGNEDBYTE</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>XSDC_DOUBLE</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>XSDC_DECIMAL</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>XSDC_DURATION</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>XSDC_DATETIMESTAMP</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>XSDC_TIME</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>XSDC_DATE</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>XSDC_GYEARMONTH</td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>XSDC_GYEAR</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>XSDC_GMONTHDAY</td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>XSDC_GDAY</td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>XSDC_GMONTH</td>
<td></td>
<td>23</td>
</tr>
<tr>
<td>XSDC_HEXBINARY</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>XSDC_BASE64BINARY</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>XSDC_ANYURI</td>
<td></td>
<td>26</td>
</tr>
<tr>
<td>XSDC_QNAME</td>
<td></td>
<td>27</td>
</tr>
<tr>
<td>XSDC_NOTATION</td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>XSDC_INTEGER</td>
<td></td>
<td>29</td>
</tr>
<tr>
<td>XSDC_ARRAY</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>XSDC_USER_TYPE</td>
<td></td>
<td>31</td>
</tr>
<tr>
<td>XSDC_NMTOKEN</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td>XSDC_ANY</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
D XSDC_NONNEGATIVEINTEGER...
D XSDC_POSITIVEINTEGER...
D XSDC_NONPOSITIVEINTEGER...
D XSDC_NEGATIVEINTEGER...
D XSDC_NORMALIZEDSTRING...
D XSDC_TOKEN...
D XSDC_LANGUAGE...
D XSDC_NAME...
D XSDC_NCNAME...
D XSDC_ID...
D XSDC_IDREF...
D XSDC_IDREFS...
D XSDC_ENTITY...
D XSDC_ENTITIES...
D XSDC_NMTOKENS...
D C_ATTACHMENT...

* ===================================================================
* Prototypes and definitions from axis/ISoapFault.h (partial)
* ===================================================================

D* ---------------------------------------------------------------
D*  Get complex fault object name
D* ---------------------------------------------------------------
D axiscSoapFaultGetCmplxFaultObjectName...
D PR              *   EXTPROC('axiscSoapFaultGetCmplxFaultObjectName')
D*                                    Returns character string
D SOAPFAULT                      *   VALUE
D*                                    ptr to soap fault object

D* ---------------------------------------------------------------
D*  Get simple fault detail
D* ---------------------------------------------------------------
D axiscSoapFaultGetSimpleFaultDetail...
D PR              *   EXTPROC('axiscSoapFaultGetSimpleFaultDetail...
9.2.6 Compiling RPG

Compile the RPG code from WebSphere Development Studio client for iSeries:

1. Expand `<your System i connection>` → iSeries Objects, right-click Library list and select Add Library List Entry.

2. In the window, enter RPGTEST in the Additional library field and click OK. This adds the RPG test library to the library list so that the compiler can access RPGINCLUDE during compilation.
3. Right-click **RPGTEST.RPGLE** and select **Compile → CRTRPGMOD** as shown in Figure 9-20.

![Figure 9-20 Compile RPG module](image)

**Note:** Errors encountered in the compile process display in the iSeries Error List. Double-clicking an error takes you to the exact position of the error in the source code.

**Important:** You can ignore all informational messages with a severity of zero (0).

### 9.2.7 Compiling C modules and Create service program

On the 5250 session, issue the following command to create the C modules and to add all modules to a single service program:

1. On the 5250 command line issue the following commands to create the C modules:

   ```batch
   CRTCMOD MODULE(RPGTEST/RPGWSC) SRCSTMF('/home/itso/RESERVEFLIGHTServices.c') INCDIR('/qibm/proddata/xmltoolkit/WSCI-1.0-os400/include')
   CRTCMOD MODULE(RPGTEST/RPGWSC1) SRCSTMF('/home/itso/RESERVEINFO.c') INCDIR('/qibm/proddata/xmltoolkit/WSCI-1.0-os400/include')
   CRTCMOD MODULE(RPGTEST/RPGWSC2) SRCSTMF('/home/itso/RESERVEFLIGHTInput.c') INCDIR('/qibm/proddata/xmltoolkit/WSCI-1.0-os400/include')
   CRTCMOD MODULE(RPGTEST/RPGWSC3) SRCSTMF('/home/itso/RESERVEFLIGHTResult.c') INCDIR('/qibm/proddata/xmltoolkit/WSCI-1.0-os400/include')
   ```
2. Issue the following command on the 5250 command line to create the Service program:

```
CRTPGM PGM(RPGTEST/RPGTEST) MODULE(RPGTEST/RPGTEST RPGTEST/RPGWSC
RPGTEST/RPGWSC1 RPGTEST/RPGWSC2 RPGTEST/RPGWSC3) BNDSRVPGM(QXMLTOOLS/QAXIS10CC)
```

### 9.2.8 Invoking the RPG application to make reservation

The RPG program calls Web Services Client for ILE Axis framework, which in turn calls the J2EE RPG Web service created in Chapter 5, “ProgramCall (RPG, Cobol) Web service” on page 35. To invoke the RPG program, run the following command from the 5250 command line:

```
CALL RPGTEST/RPGTEST
```

It should return results as shown in Figure 9-21.

<table>
<thead>
<tr>
<th>Command Entry</th>
<th>ITSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request level: 1</td>
<td></td>
</tr>
</tbody>
</table>

All previous commands and messages:

```plaintext
> call RPGTEST/RPGTEST
DSPLY The order information:
DSPLY 5671316
> call RPGTEST/RPGTEST
DSPLY The order information:
DSPLY 5671317
> call RPGTEST/RPGTEST
DSPLY The order information:
DSPLY 5671318
```

**Summary**

In the example, we demonstrated an RPG program that uses generated code from the Web Services Client for ILE to invoke a Web service that happened to be a J2EE Web service application, which encapsulates an RPG program.
JSF Web service client

JavaServer Faces (JSF) technology simplifies building user interfaces for Java Web-based applications. Developers of various skill levels can quickly build Web applications by:

- Assembling reusable UI components in a page
- Connecting these components to an application data source
- Enabling Web service and wiring client-generated events to server-side event handlers

This chapter describes how to create JSF Web services client to consume existing Web services. The JSF Web service components can go out to the Internet and search existing UDDI registries, then use the WSDL file to generate the appropriate controls. The controls use the standard JAX-RPC API to invoke the Web services.
10.1 Developing a JSF client

You can use WebSphere Development Studio client for iSeries to develop a JSF client consuming the Web service. In this section, we show you how to build a sample JSF Web service client.

These are the high-level steps to follow:
1. Create a Dynamic Web Project
2. Create Page Template
3. Create JSF Web service client using Web service Component
4. Test the JSF Web service client
5. Create Web services Proxy
6. Integrate Web services Proxy with JSF page
7. Test the JSF Web service client

We explain the steps in more detail in the following sections.

10.1.1 Creating a dynamic Web project

The first step in the development process is to create a Web project that holds all artifacts of the Web service client:

1. Open Web Perspective in WebSphere Development Studio client for iSeries:
   b. In the workspace launcher window, for workspace enter c:\temp\redbook and click OK.
   c. To open Web perspective, select Window → Open Perspective → Other, select Web and click OK.
2. Create a Dynamic Web Project:
   b. Enter the Project Name: JSFWSclient.
   c. Click the Show Advanced button and make sure that the EAR project name is JSFWSclientEAR and that the context root is the same as the project name.
   d. Click Finish.
3. In the Project Explorer view, expand Dynamic Web Projects → JSFWSclient project → WebContent.

Before you create any Faces JSP pages, you create page template that you will use for the JavaServer Faces.

10.1.2 Creating a page template

In this section, we explore how to create a page template that you can used for the JSF pages. Before you create the page template, you need to create or import the necessary images. WebSphere Development Studio client has a built-in Web art designer tool that you can use to create the images and banner. Let us look at how to create an image file.
Creating an image file
To create image file using the Web Art Designer tool:
1. From menu select **File → New → Image file**.
2. Enter the filename `flight400banner` and click **Finish**.
3. Double-click `flight400banner` under WebContent folder to open it in WebArt designer tool.
4. In WebArt designer tool window, select **Edit → Canvas settings**.
5. Enter the following values:
   - Enter Width: 640
   - Enter Height: 79
6. Click OK.
7. In WebArt designer tool, click the Gallery tab (see Figure 10-1).

![Figure 10-1 Image creation - Gallery tab](image-url)
8. Click **Banners** to see the banner list. Double-click the desired banner image from the banner list view (see Figure 10-2).

![Figure 10-2 Image creation - Select banner image](image-url)
9. Next, you create a logo and place it on the banner. Click the WebArt Gallery tab. Enter the desired text in the Text input box as shown in Figure 10-3.

![Figure 10-3 Image creation - banner text]

10. Double-click desired text format.

11. Resize the logo text as desired.

![Figure 10-4 Image creation - text format]

12. Select File → Save Canvas or press CTRL+S to save the JPG.

13. On the Save Canvas panel, click Save.

14. On the JPEG attribute settings window, click OK.

15. Select File → Exit to exit the WebArt Designer tool.
**Importing required images**
You can add additional images to your page template. Follow these steps:

1. Make sure **WebContent → Theme** folder is selected.
2. Select **File → Import** from menu. In the Import window, select **File System** and click **Next**.
3. Navigate to the location of your image files.
4. Select the desired files. (We select `plane.jpg` in this example.) Make sure that the Into folder points to the JSFWSclient/WebContent/theme folder.
5. Click **Finish**.

**Creating page template**
Now you have all art artifacts to create a page template. Follow these steps:

1. Make sure that the WebContent\theme folder is selected under the JSFWSclient Project.
2. Select **File → New → Page Template File**.
3. In the New Page Template File window, enter **Flight400PageTemplate** as the File Name and select **Template Containing Faces Components** in the Model drop-down box.
4. Click **Finish**.
5. Click **OK** on any informational window that displays.
6. In Page Template editor view, delete the default text *Place content here*.
7. In Page Template editor view, select **Free Layout** mode (see Figure 10-5).

8. Open the HTML tags drawer in the palette view and drop **Free Layout Table** (see Figure 10-6). Free Layout tables create tables and cells for you automatically so that you can freely place objects on the page.
9. Drag the corners of free layout table and increase the size.

10. Select File → Save or press CTRL+S to save your work.

11. Select `plane.jpg` from the WebContent → theme folder and drag-and-drop it in the first row of the free layout table as shown in Figure 10-7.

![Figure 10-7   Page template - insert image](image)

12. Open the Web Site Navigation drawer under Palette. Drag Navigation Trail and drop it in the second row on the right-hand side as shown in Figure 10-8.

![Figure 10-8   Page template - navigation trail](image)

13. In Select a navigation specification file, select trail_horizontal.jsp and click Finish.
14. Select the cell where navigation trail is placed. Use the guides to expand the cell across the layout area (see Figure 10-9).

![Figure 10-9  Page template - expand navigation trail](image)

15. Drag **Vertical Tabs** from the Web Site Navigation drawer under Palette and drop it below the plane image (see Figure 10-10).

![Figure 10-10  Page template - vertical tabs](image)

16. In Select a Navigation Specification file window click **Next**.

17. In Select Links Destination window:
   a. Select **Children of top page** so that your navigation bar has links to the child pages of the home page.
   b. Then select **Sibling pages** to create navigation links between pages at the same level in the site hierarchy.
   c. Click **Finish**.
18. Select the cell where Vertical Tabs is placed. Use the guides to expand the cell across the next column and rows below as shown in Figure 10-11.

![Figure 10-11  Page Template - Vertical Tabs expanded](image)

19. Open Page template drawer. Drag **Content Area** and drop it below the navigation trail (see Figure 10-12).

20. In the Insert Content Area pop-up click **OK**.

![Figure 10-12  Page Template - Content Area](image)
21. Expand the content area using guides to occupy next column and row below as shown in Figure 10-13.

![Figure 10-13  Page Template - Content Area Expanded](image)

22. Select **File → Save** or press CTRL+S to save your work.
23. Select **flight400banner.jpg** from the WebContent folder in Project Explorer. Drag and drop it in the first row next to plane.jpg (see Figure 10-14).

![Flight400PageTemplate.jplx](image)

*Figure 10-14  Page Template - banner image*

24. Select **File → Save** or press CTRL+S to save your work.
25. Change the background color. Select the left navigation pane in the page template. Choose the desired color in the Color field in the properties view (see Figure 10-15).

26. Select **File → Save** or press CTRL+S to save your work.
27. Preview your page template by selecting the Preview tab (see Figure 10-16).

28. Select File → Close to close the page template editor window.

10.1.3 Creating JSF Web service client using Web Service Component

In this section, you use the page template and existing Web service that you created earlier to create a JavaServer Faces Web service client. You use some of the JSF editing features to drag-and-drop components.

Reviewing a WSDL document

Verify that you still can access a Web service:

1. Type the path for the WSDL document URL in an Internet Explorer window and press Enter. In our example, the URL is:

   http://<System_i>:<port>/WebServiceProject/wsd1/com/ibm/flight400/beans/GETFLIGHTINFOServices.wsdl

   You should see the content of the WSDL file.

2. The file shows different methods implemented within the Web service with the input and output parameters. In our example, we have two methods: findFlights and findFlights_XML. The methods have three input parameters (FROMCITY, TOCITY, and FLIGHTDATE) and two output parameters (FLIGHTCOUNT and FLIGHTS data structure).

3. We use this WSDL document to build the JSF Web service client to consume the Web service.
Creating a Faces JSP page

It is time to build a JSF based on the page template that you developed earlier in this chapter. Follow these steps:

2. From the menu bar select File → New → Faces JSP file.
3. Enter the filename findFlightsJSFclient.
4. Select Create from page template and click Next (see Figure 10-17).

![Figure 10-17  Faces JSP properties]
5. On Page template file selection panel, select **User-defined page template**, select the previously created **Flight400PageTemplate**, and click **Finish** (see Figure 10-18).

![New Faces JSP File](image)

**Figure 10-18  User-defined page template**

6. Select the default text **Default content of bodyarea** and replace it with **Flight Search Web service client**.

7. Press Enter next to Flight Search Web service client.

8. Select **Flight Search Web service client** and click the Properties tab at the bottom of the window.
9. The properties view shows the properties for the selected component in the edit window. Select **Heading 1** in paragraph field (see Figure 10-19). You can also change the color and the font. Press the down arrow to keep the cursor below the heading.

![Figure 10-19   Page Heading](image)

10. Save your work by selecting **File → Save** from the menu bar or by pressing CTRL+S on the keyboard.

11. Make sure your cursor is just below the heading. In the **Palette** view (on the right-hand side), expand **Data drawer**.
12. From Data drawer, select Web service and drop it onto the JSF page at the cursor position (see Figure 10-20).

13. The Web services discovery dialog box displays. Click the Web services from a known URL link. Also, you can search Web services deployed and running in your current workspace.

14. Enter the WSDL URL (see step 1 on page 231) and click Go.

15. You should see Web services details with port information. To explore the Web service click Details. This launches Web Services Explorer in the browser. You should see the methods and endpoint information. Click findFlights to test that method. On Invoke a WSDL operation screen, enter following inputs:
   - FROMCITY: Albany
   - TOCITY: Atlanta
   - FLIGHTDATE: 09122006
16. Click **Go**. It shows the result in the Status pane (Figure 10-21).

![Figure 10-21 Web services Explorer](image)

17. You can also test the `findFlights_XML` method to see the output in XML document form. Close the Web services Explorer browser window.

18. Back in Web Services Discovery dialog box, click **Add to Project**.

19. Click the **Yes All** button on the Warning dialog box.

20. Select the `findflights()` method on the Web services wizard page. Make sure that **Create input form and result display** selected. Click **Next** (see Figure 10-22).

![Figure 10-22 Select a method](image)
21. On the Input form page, it shows the input fields that are placed on the Input form. Click **Options**. Enter *Search Flights* in the Submit button field and click **OK** (see Figure 10-23). The options dialog lets you select input field properties for the Input form.

![Figure 10-23  Input Form - options](image)

22. Back in Input Form dialog box, click **Next**.
23. The Result form shows the output fields. Click **Finish**.
24. JSF Web services wizard generates input and output fields on the JSF page as shown in Figure 10-24.

25. Save your work by selecting **File → Save** in the menu bar or by pressing CTRL+S on the keyboard.
26. Update labels next to the input fields to make them readable. You can also select the text and change the properties like format, font, color, and so on in the Properties view. Update the labels on the output table. Select the column heading and change the value in the properties view. The updated page is shown in Figure 10-25.

![Figure 10-25 Input and Output fields](image)

27. Save your work by selecting **File → Save** from the menu bar or by pressing CTRL+S on the keyboard.

Now you are ready to test the Web service client in WebSphere Application Server Test environment in WebSphere Development Studio client for iSeries.

### 10.1.4 Testing the JSF Web service client

To test the JSF Web service client:

1. Right-click **findFlightsJSFClient.jsp** in the Project Explorer view under **JSFWSclient → WebContent** folder and select **Run → Run on Server** option.

2. In the Server selection window, click **Choose an existing server** and select **WebSphere Application Server V6.0** under localhost and click **Finish**.

3. After a while, you should see a browser displaying the findFlights page with input form. Enter following inputs:
   - From City: Albany
   - To City: Atlanta
   - Date: 09122006
   - Click Search Flights
4. Double-click a Web browser window title to expand the window (see Figure 10-26).

![Figure 10-26 Test the JSF Web service client](image)

**10.2 Summary**

In this chapter, we demonstrated an example of building a JSF Web service client. WebSphere Development Studio client for iSeries provides a great deal of tools to make this task as easy as possible. You can now design, build, and test a JSF client without leaving the WebSphere Development Studio client for iSeries environment.
PHP Web service client

In this chapter, we describe how to consume Web services using the scripting language PHP. The three examples demonstrate Web services clients from simple to more complex. The first two examples are the clients for the Web services created in Chapter 8, “PHP Web service” on page 171, and the third is a client that is designed to consume the GETFLIGHTINFOServices Web service that you built in 5.2.14, “Adding additional Web services: GetFlightInfo and FindCustomers” on page 70.
11.1 Consuming a Web service with PHP

This section explains how to consume the following Web services:

- The first Web service, *Repeater*, is a PHP Web service with two functions. It takes in a string and an integer. The function `repeat` repeats a string of text a number of times and the function `reverseRepeat` reverses a string of text and repeats it a number of times (see Example 8-8 on page 182).

- The second Web service, *ProgramCall*, uses a PHP Web service with one function. It takes first name, last name, and an initial amount as input. The Web Service wraps an `i5_program_call` to the INCRMT CL program that sets the account name to the last name, increments the balance by specified amount, and returns the results (see Example 8-11 on page 188).

- The third Web service *GETFLIGHTINFOServices* is a WebSphere Web service that wraps FLGHT400M/NSF404(FINDFLIGHTS). It takes the departure and destination cities and the date of the flight as input. It returns the number of available flights and available flight information.

11.1.1 Using Zend Studio for i5/OS to create Web services clients

Zend Studio for i5/OS can simplify the task of writing PHP Web services clients. For the examples in this chapter, we use the PHP SOAP extension to create the Web services client. To create the SOAP Client, you need the WSDL file. It can reside on a system where you run a PHP client, you can specify it as a URL.

After you create the SOAP client, two useful features become available in Zend Studio for i5/OS. The first is the *Inspectors* view of available SOAP clients. The new SOAP Client is visible in the File and Project Inspector view. With this view, you can expand the SOAP client to reveal the name of the WSDL, the class it extends, the available functions, what they take for input, and what they return for output. The second feature that is available in Zend Studio for i5/OS after you create the SOAP client is the code completion feature. It analyzes dynamically the WSDL file that is associated with the SOAP client and displays information that is relevant to the function as you type.
Figure 11-1 illustrates both features.

The technique for creating a PHP client code in Zend Studio for i5/OS is the same as for creating a PHP server code. You need to have a project to which you add your PHP files. Use the File → New File menu option to create a new PHP file.

11.1.2 Consuming the Repeater PHP Web service

Example 11-1 was created using Zend Core Version 1.6.0, PHP Version 5.1.6. In this first example, we show you how to consume the PHP Repeater Web service (see Example 8-8 on page 182). Notice that near the bottom of repeaterClient.php, there is the optional code to print the last SOAP request and response. It is not commented out so when the code is run in addition to the Web service reply, the actual last SOAP request and response is displayed also.

Example 11-1 repeaterClient.php source code

```php
<?php
try {
    /* Create New Soap Client */
    $client = new SoapClient("repeater.wsdl",array(
        "trace" => 1,
        "exceptions" => 0));

    // Call repeat function:
    echo $client->repeat("Hello ","<hr />");

    // Call reverse repeat:
    echo $client->reverseRepeat();

    // Optional Code to print last SOAP Request and Response:
    try {
        print "<p>Call repeat function:

        " . $client->__getLastRequest() . "

        " . $client->__getLastResponse() . "</p>"
    } catch (SoapFault $fault) {
    } catch (Exception $exception) {
        print "Exception: " . $exception->getMessage() . " (" . $exception->getCode() . ")"
    }
}
```
/** Values are hardcoded in this example but they could easily be input from the user **/

/* Call repeat function*/
   echo $client->repeat("Hello ",7)."<br />
";

/* Call reverse repeat function*/
   echo $client->reverseRepeat("My name is Michael",3);

}catch (SoapFault $exception) {
   echo $exception;
}

/** Optional Code to print the last SOAP Request & Response **/
print "<p>
print "<b>Request :</b>
htmlspecialchars($client->__getLastRequest()) ."\n";
print "</p>";
print "<b>Response :</b>
htmlspecialchars($client->__getLastResponse())."\n";
print "</p>";
?>

Figure 11-2 shows a sample output from the repeaterServer Web service.

Figure 11-2  The repeaterClient.php run in Web browser
11.2 Consuming the ProgramCall PHP Web service

Example 11-2 was created using Zend Core Version 1.6.0, PHP Version 5.1.6. This example shows how to consume the PHP ProgramCall Web service (see Example 8-11 on page 188). This Web service client is more complex than Example 11-1 in that it works with the complex data type returned by the Web service, working with it as an object.

Example 11-2  programCallClient.php source code

```php
<?php
/**
 Values are hardcoded in this example but they could easily be input from the user **/
$first = "Michael";
$last = "Sandberg";
$amount = "10";

/** Create SOAP Client **/
try {
    $client = new SoapClient("programCall.wsdl", array(
        "trace" => 1,
        "exceptions" => 0));

    print "<p>
    $results = $client->callINCRAMT($first,$last,$amount);
    /** Optional Code to print the $results object **/
    //var_dump($results);
    print "</p>";
} catch (SoapFault $exception) {
    echo $exception;
}

echo "First and last name and an initial amount are passed into the Web Service. The Web Service wrappers the i5_program_call to the INCRAMT CL program that sets the account name to the last name, increments the amount entered by 42.22, and returns the results. The Web Service then passess the results back to the Web Service Client.";

echo "<br>
First Name: $first<br>
Last Name: $last<br>
Initial Amount: $amount<br><br>
Results from Web Service call:<br>

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Account Balance</th>
<th>Account Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$first</td>
<td>$last</td>
<td>$amount</td>
<td>$account</td>
</tr>
</tbody>
</table>

/** Optional Code to print the last SOAP Request & Response **/
//print "<p>
//print "Request :".htmlspecialchars($client->__getLastRequest())."\n";
```
Figure 11-3 shows a sample output from the programCall PHP Web service.

Figure 11-3 The programCallClient.php run in Web browser

11.2.1 Consuming the GETFLIGHTINFOServices WebSphere Web service

Example 11-3 was created using Zend Core Version 1.5.0, PHP Version 5.1.6. It shows how to consume the GETFLIGHTINFOServices WebSphere Web service (see 5.2.14, “Adding additional Web services: GetFlightInfo and FindCustomers” on page 70). This example is the most complex of the three examples in this chapter. The Web service client passes in a complex data type and also receives back another complex data type. To pass in the required parameters to the Web service, an object is created that matches the design of the expected complex data type. The results are passed back as an object with a variable number of flights. This is handled by retrieving the number of flights value from the results object and then using it in a for loop to walk through the results object printing the values.

Also notice that this example points to the WSDL file that is located on a remote server (the highlighted text in the example). We use the file's URL. Another way of using remote WSDL file is to copy it to the system where you run your PHP client code. In the latter case, you would point to the WSDL file in the same way as we show in the Example 11-1 and Example 11-2.

Example 11-3 WSClent.php source code

```php
<?php
/**
 * WSClent is a PHP Web services client designed to consume the Web service described
 * by GETFLIGHTINFOServices.wsdl
 **/

/** Build the Object that will be passed in the call to the Web service
 * Values are hardcoded in this example but they could easily be input from the user
 **/
$params -> inputData -> FROMCITY = 'Chicago';
$params -> inputData -> TOCITY = 'Atlanta';
```
```php
// Create SOAP Client
try {
    $client = new SoapClient("http://remoteServer:9080/WebSvc/wsd1/com/ibm/flight400/beans/GETFLIGHTINFO\Service.wsdl", array(
        "trace" => 1,
        "exceptions" => 0,));
} catch (SoapFault $exception) {
    echo $exception;
}

/** Call Web service described in GETFLIGHTINFO\Service.wsdl passing $params **/
try {
    $results = $client->findflights($params);
    /** Optional Code to print the $results object **/
    //print "<p>
    //var_dump($results);
    //print "</p>";
} catch (SoapFault $exception) {
    echo $exception;
}

$param = new stdClass();
$param->inputData = new stdClass();
$param->inputData->FROMCITY = 'New Delhi';
$param->inputData->TOCITY = 'Mumbai';
$param->inputData->FLIGHTDATE = '07122007';

// Create SOAP Client
try {
    $client = new SoapClient("http://remoteServer:9080/WebSvc/wsd1/com/ibm/flight400/beans/GETFLIGHTINFO\Service.wsdl", array(
        "trace" => 1,
        "exceptions" => 0,)));
} catch (SoapFault $exception) {
    echo $exception;
}

/** Call Web service described in GETFLIGHTINFO\Service.wsdl passing $param **/
try {
    $results = $client->findflights($param);
    /** Optional Code to print the $results object **/
    //print "<p>
    //var_dump($results);
    //print "</p>";
} catch (SoapFault $exception) {
    echo $exception;
}
```

```
Before attempting to execute the PHP client code, make sure that your Web service on the remote system has been started. Figure 11-4 shows sample output from the PHP Web service client.

![Figure 11-4  WSClient.php run in browser](image-url)
Setting the connection to WebSphere Application Server V6.0

You can use the SOAP connector or remote method invocation (RMI) to make JMX™ connections with the server. These ports are used for communication between the development environment and the server. The RMI (ORB bootstrap) port is designed to improve performance and communication with the server. The SOAP connector port is designed to be more firewall compatible. It is used by an HTTP transport for incoming SOAP requests.

This chapter explains how to set the connection to WebSphere Application Server V6.0 using a JMX option.

**Important:** If there is a firewall between the development environment and the server, use the SOAP connector port rather than the ORB bootstrap port.

**Important:** In a lab environment where a single image is ghosted to multiple PCs, the master image host name is ghosted to the WebSphere Application Server - Test Environment runtime configuration files. To resolve the TCP issue, use SOAP.
Changing a JMX connection with a server

Follow these steps to view and change, if needed, a JMX connection to your server in WebSphere Development Studio client for iSeries:


   **Note:** In the event that WebSphere Development Studio client for iSeries is already started, proceed to step 4 on page 251.

2. In the workplace launcher window, enter c:\temp\redbook in the workspace field and press OK as shown in Figure A-1. (Make sure that you deselect the check box **Use this as a default and do not ask again.**)

   ![Figure A-1 Specifying a workspace location](image)

3. If the WebSphere Development Studio client for iSeries workspace was not used before, you see a Welcome page. Click X to close the Welcome page (see Figure A-2).

   ![Figure A-2 Closing a Welcome page](image)
4. Click the **Open perspective** icon on the right-hand side (see Figure A-3) or go to **Window → Open Perspective**.

![Figure A-3](image)

5. Select **Other**.

6. In the Select Perspective window, select **Web** and click **OK**.

7. In Web Perspective, select the **Servers** view as shown in Figure A-4.

![Figure A-4](image)
8. In the Servers view, double-click **WebSphere Application Server v6.x** to open the server editor (see Figure A-5).

![Figure A-5  Double-clicking your server](image)

9. Click the Overview tab. Expand the **Server** section if it is not expanded already.

10. Under the Server connection type and admin port, select **SOAP (more firewall compatible)** (see Figure A-6). For a SOAP connection, in the SOAP connector port field, we use the default port number **8880**.

![Figure A-6  Selecting the SOAP connector](image)

11. Select **File → Save** from the menu bar or press CTRL+S to save the configuration changes.
12. Click X to close the WebSphere 6.0 Server Configuration view (see Figure A-7).

![Figure A-7  Closing the server properties file](image)

13. In the Servers view, right-click the WebSphere 6.0 server (you might see a different name in your environment) and select Start (see Figure A-8).

![Figure A-8  Starting the server](image)
14. In the Servers view, you should see the *Starting* message for 3 to 5 minutes as shown in Figure A-9.

![Figure A-9](image1.png)

*Figure A-9  The server is starting*

15. After the WebSphere Application Server is started, you should see the *Started* status in the Servers view as shown in Figure A-10.

![Figure A-10](image2.png)

*Figure A-10  The server is started*
Appendix B. URI length limit of 259 characters on Windows

This appendix describes the details of the workaround for WebSphere Development Studio client for iSeries in a case when a path to a resource in WebSphere Development Studio client for iSeries exceeds 259 characters.
Issue

An error such as that shown in Example B-1 can occur when you are deploying an application to the WebSphere Test Environment using WebSphere Application Server V6.0.

Example: B-1   Error message

[11/14/05 13:04:26:114 EST] 00000042 SystemErr R java.io.IOException: URI length is greater than Windows limit of 259 characters. C:\Program Files\IBM\Rational\SDP\6.0\runtimes\base_v6\profiles\default1\wstemp\1079040961d\workspace\cells\sandygmoNode05Cell\applications\MyProjectWithaLongPathNameEAR.ear\deployments\MyProjectWithaLongPathNameEAR\MyProjectWithaLongPathNameWeb.war\wssl\a
very\exceptionallylong\nameusedforthepackagename\MyTestForWS.wsdl

Cause

The length limit on Windows is imposed by the Java SDK 1.4.

Solution

Here are a few different suggestions to try for resolving this issue:

- Shorten the names of the Enterprise Application Project (EAR) and Web Project
- Shorten the length of the package used
- Shorten the temp directory used by WebSphere Application Server by following the steps we describe in this section.

**Important:** In this section, the solution to shorten the temp directory is shown in detail for WebSphere Development Studio client for iSeries, because this is the most complete resolution to the problem.

Follow these steps to resolve the issues with a path length:

1. Create a directory with a short name, for example C:\A, on the C drive. Using Windows Explorer select File → New Folder as shown in Figure B-1.

![Creating a new directory](image)

3. In the workplace launcher window, enter `c:\temp\redbook` in the workspace field and click **OK** (see Figure B-2).

**Note:** In the event that WebSphere Development Studio client for iSeries is already started, proceed to step 5 on page 257.

![Figure B-2 Specifying a workspace location](image)

4. If you have not used WebSphere Development Studio client for iSeries before, you should see a Welcome page. Click **X** to close the Welcome page as shown in Figure B-3.

![Figure B-3 Closing a Welcome page](image)

5. Click the **Open perspective** icon on the right-hand side (see Figure B-4) or go to **Window → Open Perspective**.

![Figure B-4](image)

6. Select **Other**.

7. In the Select Perspective window, select **Web** and click **OK**.
8. In the Servers view, right-click the **WebSphere 6.0 server** (you might have a different name in your environment) and select **Start** as shown in Figure B-5.

![Figure B-5   Starting the server](image)

9. In the Servers view, you should see the Starting message for 3 to 5 minutes (see Figure B-6).

![Figure B-6   The server is starting](image)
10. After the WebSphere 6.0 Server is started, you should see the *Started* status in the server view (see Figure B-7).

![Figure B-7](image)

*Figure B-7  The server is started*

11. In the Server view, right-click the **WebSphere 6.0 server** and select **Run administrative console** (see Figure B-8).

![Figure B-8](image)

*Figure B-8  Opening the administrative console*
12. Double-click **Admin Console** window to expand it (see Figure B-9).

![Figure B-9 Expanding the window](image1)

13. In the Admin Console window, enter a user name (any name will work) and click **Log in** (see Figure B-10).

![Figure B-10 Logging in](image2)
14. Expand **Servers** by clicking the plus (+) sign and select **Application Servers** (see Figure B-11).

15. Select **server1** under Application Servers as shown in Figure B-12.
16. Expand **Java and Process Management** by clicking the plus sigh (+) under Server Infrastructure. Click **Process Definition** (see Figure B-13).
17. Select **Java Virtual Machine** under Additional Properties as shown in Figure B-14.
18. In the Generic JVM arguments, enter `-Dworkspace.user.root=C:/a` and click **Apply** as shown in Figure B-15. The directory you specify must match the directory that you created in step 1 on page 256.

![Figure B-15  Changing JVM arguments](image)

19. Click **Save** on the next window to save the changes (see Figure B-16).

![Figure B-16  Saving WebSphere profile configuration](image)
20. Click **Save** on the next panel (see Figure B-17).

![Figure B-17](image1)

21. Click **Logout** to exit the Admin Console window as shown in Figure B-18.

![Figure B-18](image2)

22. Click **X** to close the Admin Console window (see Figure B-19).

![Figure B-19](image3)
23. In the Servers view, right-click **WebSphere 6.0 server** and select **Restart → Start** (see Figure B-20).

![Figure B-20  Restarting the server](image)

24. Wait until the server status changes to **Started** (see Figure B-21).

![Figure B-21  The server is restarted](image)
Useful tools

This chapter examines some of the advanced topics for application debugging on System i. It includes the following topics:

- SOAP monitoring utility
- Security testing for Web services
- WebSphere Development Studio client for iSeries debugger
**SOAP monitoring utility**

In the previous chapters, we showed how to generate the Service Provider or Service Consumer. While testing the interaction, nothing happens between the services or, worse yet, you get exceptions or Web services faults. In this section, we articulate a sequence of steps to help with problem determination of the service interaction. This problem determination should prove helpful for both the Service Consumer or Service Provider scenarios.

In the course of the development of Web services, it became apparent that seeing the messages exchanged between the Service Provider and Service Consumer is necessary to assist in debugging technical problems.

There are several SOAP utilities that you can use to monitor the messages between service provider and service consumer. In the next sections, we focus on the TCPMON utility offered as a free download from:

http://www.apache.org

You can use the TCPMON utility for the following issues:

- Response timeout
- Web service faults that seem network related
- Service end-point interface that seems incorrect
- Examine the SOAP message being exchanged between the Consumer/Provider
- Determine if encryption of messages are occurring within the SOAP packet

**Download Apache TCPMON binary**

TCPMON is a java based utility that can be downloaded from the following location:

http://ws.apache.org/commons/tcpmon/download.cgi

You do not need to run the utility on the Consumer or the Provider. You can run it on any GUI-based client and use it as mediator to examine the exchanging of the messages. TCPMON requires a version of JDK™ to run the utility. Follow these steps:

1. Extract the contents of the compressed file to a temporary directory.
2. Select to execute ..\tcpmon-1.0-bin\build\tcpmon.bat (see Figure C-1).

![TCPMON Utility](image1)

**Figure C-1  TCPMON Utility**

3. Click the Admin tab. Enter the values Target Hostname and Target Port # which can be determined from the WSDL file (For further information about Service Endpoint review 5.2.9, “Modifying the Web service Client URI” on page 62). Click **Add** (see Figure C-2).

![TCPMON Admin tab](image2)

**Figure C-2  TCPMON Admin tab**
4. Select the new listener tab (see Figure C-3) that was created to monitor SOAP packets as they are routed through the TCPMON SOAP utility.

![TCPMON SOAP Listener](image)

Figure C-3  TCPMON SOAP Listener

5. Adjust any Web service client service endpoint interface to point at the listener port and server that you defined in the previous step. (You should use the host name of the system where you run TCPMON and the port number that you specify on the Admin tab. In our example, it is 10480.)

- JSP test client - 10.1.3, “Creating JSF Web service client using Web Service Component” on page 231
- RPG Client - 9.2.2, “Examining the WSDL document” on page 198
- Web Services Explorer on WebSphere Development Studio client for iSeries as shown in the next steps
6. Using WebSphere Development Studio client, right click any WSDL document and select **Web services → Test with Web services Explorer** (see Figure C-4).

![Figure C-4 Web services Explorer](image1.jpg)

7. On Web services Explorer select **Add** (see Figure C-5).

![Figure C-5 Web services Explorer](image2.jpg)
8. Enter the URL to route through TCPMON Listener and click **Go** (see Figure C-6).

![Configure TCPMON as EndPoint for Web service](image)

**Figure C-6** Configure TCPMON as EndPoint for Web service

9. Select the Web service operation in the Operations tab as shown in Figure C-7.

![Service Operations](image)

**Figure C-7** Service Operations

10. Enter the appropriate values as required for your Web service operation and click **Go** (see Figure C-8).

![Invoking Web service](image)

**Figure C-8** Invoking Web service
11. Examine the TCPMON Trace for details (see Figure C-9).

TCPMON uses in debugging services problems

A TCPMON trace information is actually hitting the wire, and this is helpful if you are unsure whether the service client is getting to the provider. It also responds back through the TCPMON utility indicating the complete flow of Web service interaction.

TCPMON can also be used in a case where some Service Clients are working and other Service Clients are not working. For example, say a new RPG Web service Consumer was generated with the IBM Web Services Client for ILE and the generated code is not working. However, the JSP or Web services Explorer client is working successfully. Capturing TCPMON packets for both clients might indicate why one service is working and the other service is failing.

TCPMON uses for security

Security is a concern in an SOA implementation. Many times message and information within the message might need to be encrypted or authenticated. Much the security discussions are beyond the scope of this book. However, if you need to verify that your message has been encrypted, use the same steps as described in “TCPMON uses in debugging services
problems” on page 273. You can view the packets to determine whether the information was indeed encrypted.

**WebSphere Development Studio client - debugger**

In the development of existing applications including RPG and COBOL acting as service provider or service consumer a useful tool is the debugger built into WebSphere Development Studio client. The debugger allows you to step line by line through the RPG code as it is interacted within a services model.

*Note:* In this section we use RPGCODE/RPGTEST program to demonstrate debugging technique. You need to replace this name with your program’s name as you perform debugging activity.

**Using the WebSphere Development Studio client debugger**

In this section, we demonstrate how to use the host application debugger. We show this using the Web services client RPG application that we described in Chapter 9, “IBM Web Services Client for ILE (RPG, C, C++, COBOL)” on page 195, a service client written in RPG was demonstrated in detail. Follow these steps:

1. Open the Remote Systems Explorer perspective with **Window → Open Perspective → Other** (see Figure C-10).

![Figure C-10 Switch perspective](image)
2. Select **Remote System Explorer** and click **OK** (see Figure C-11).

![Remote System Explorer](image1)

*Figure C-11  Remote System Explorer*

3. Expand **System i → iSeries Objects**, right-click **Library list**, and select **Add Library List Entry** (see Figure C-12).

![Add Library List](image2)

*Figure C-12  Add Library List*
4. On the Add Library List Entry frame, enter the library of source code in the Additional Library field (see Figure C-13) and click OK.

![Figure C-13   Library List Entry](image)

5. In Remote Systems Explorer frame expand into the library that contains, in our example, the QRPGLESRC source file. This file includes the RPG source code member.

6. Right-click the RPG source member and select Compile (Prompt) → CRTRPGMOD (see Figure C-14).

![Figure C-14   Compile RPG](image)
7. Select *ALL in the Debugging views field and click **OK** (see Figure C-15).

![Create RPG Modulo (CRTFRPGMOD)](image)

*Figure C-15 Include Debug Code*

8. Repeat the steps 9.2.7, “Compiling C modules and Create service program” on page 217 to complete the generating the client code.

9. Using WebSphere Development Studio client, right-click the source code library and select **Refresh**.
10. Right-click **RPGTEST** → **Debug (Service Entry)** → **Set Service Entry Point** (see Figure C-16).

![Figure C-16  Set Service Entry Point](image)

**Attention:** A warning might appear indicating that you need to start the debug server on the System i host (Figure C-17). Open a 5250 session your System i platform and issue the command STRDBGSVR. After Debug Server starts repeat the previous step.

![Figure C-17  STRDBGSVR](image)

11. On the 5250 panel in which the debug server is running, issue the command CALL RPGTEST/RPGTEST.
12. Immediately switch back to WebSphere Development Studio client. The application will switch to the debugger perspective after the application is called from 5250 (see Figure C-18).

![Figure C-18  Debugger for System i - RPG](image)

13. Use the buttons shown in Figure C-19 to debug the application (there are Play, Terminate, Step Into, Step Over, and Step Return buttons).

![Figure C-19  Debugger buttons](image)
Additional material

In this appendix, we describe the additional material that we refer to in this book which you can download from the Internet.

Locating the Web material

The Web material that is associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

ftp://www.redbooks.ibm.com/redbooks/SG247284

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 IBM Redbooks form number, SG247284.

Using the Web material

The additional Web material that accompanies this book includes the following files:

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flight400.zip</td>
<td>zipped 2 SAVF files</td>
</tr>
</tbody>
</table>

How to use the Web material

Create a subdirectory (folder) on your workstation, and decompress the contents of the Web material zipped file into this folder. FTP the two SAVF files to your i5/OS server, using binary transfer.

Use RSTLIB commands on your i5/OS server to create two libraries:

```c
RSTLIB SAVLIB(FLGHT400) DEV(*SAVF) SAVF(<your LIB name>/FLGHT400)
RSTLIB SAVLIB(FLGHT400M) DEV(*SAVF) SAVF(<your LIB name>/FLGHT400M)
```
Related publications

We consider the publications that we list in this section particularly suitable for a more detailed discussion of the topics that we cover in this book.

IBM Redbooks publications

For information about ordering these publications, see “How to get IBM Redbooks publications” on page 284. Note that some of the documents that we reference here might be available in softcopy only.

- Web Services Handbook for WebSphere Application Server 6.1, SG24-7257
- WebSphere Version 6 Web Services Handbook Development and Deployment, SG24-6461
- Patterns: SOA Foundation Service Connectivity Scenario, SG24-7228
- Patterns: SOA Foundation Service Creation Scenario, SG24-7240
- Patterns: SOA Foundation - Business Process Management Scenario, SG24-7234
- Enabling SOA Using WebSphere Messaging, SG24-7163
- Patterns: SOA Client - Access Integration Solutions, SG24-6775

Other publications

These publications are also relevant as further information sources:

- Real World Web services, by Will Iverson, ISBN 0-596-00642-X (O'Reilly Media)

Online resources

These Web sites and URLs are also relevant as further information sources:

- SOA introduction
- developerWorks: SOA and Web services
  http://www-128.ibm.com/developerworks/webservices
- Wikipedia: Service-oriented architecture
- System i Developer Roadmap: End to End Demo
  http://www-03.ibm.com/servers/enable/site/ideveloper_j2ee/etoe/index.html
WebFacing Deployment Tool with HATS Technology

Web Services Client for ILE introduction

PHP Web site
http://www.php.net/

i5/OS PHP Enabling Technology
http://www.zend.com/products/zend_core/zend_for_i5_os

How to get IBM Redbooks publications
You can search for, view, or download IBM 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

Symbols
.Net 7

Numerics
5250 session 155
5250W 129
5722-JC1 35
5722JC1 39
5722JV1 39
5722SS1 39
5722WDS 39
5733XT1 196

A
adapter 15
Analyze the existing application 36
Apache 173
Application Modernization 36
architecture 4
asynchronous communication 21
asynchronous Web service 15
Axis SOAP engine 21

B
Basic Profile 24
Basic Profile standard 21
BPEL 15
broker 24
Business driven reasons to consider SOA 8
business needs 8
Business Process Execution Language 15
business requirement 12
business task 12

C
callable functions 37
Changing a JMX connection with a server 250
Choreography Description Language 15
COBOL 196
com.ibm.as400.access.AS400JDBCDriver 87
complex data type 188
data macro 130
direct JDBC™ connection 77
Consuming the GETFLIGHTINFO Services WebSphere Web service 246
Consuming the ProgramCall PHP Web service 245
CORBA 7
Creating a DB2 stored procedure 95
Creating a dynamic Web Project 80
Creating a Faces JSP page 232
Creating a HATS project 126
Creating a PHP Web service to wrapper a program call 187
Creating a simple Web service from PHP 182
Creating a Web service with PHP 180
Creating an SQL statement 84
Creating and building an SQL stored procedure 96
Creating DB2 UDB Web services 75
Creating page template 224
Creating the DADX file 87
Creating the DADX file and generate the Web service 104
Creating the integration object 158
CRTCMOD 217
Cumulative Group PTF 39

D
DAD 78
DADX 78
DADX approach 78
DADX file 79–80
call to a stored procedure 79
delete 79
insert 79
query 79
update 79
DADX group 78
DAMP 173
data macro 130
Database Server 81
DB2 connection setup 83
DB2 system 86
DB2 Web services architecture 77
DB2 Web services solution 77
DB2 XML Extender 77–78
query 78
Storage 78
Defining a connection to the i5/OS server 41
Deploying a Web service to WebSphere Application Server for i5/OS 61
Deploying the Web services 104
Designing an SOA solution 12
Developing a DB2 UDB Web service 80
Developing a HATS Web service 126
Developing a JSF client 220
direct JDBC™ connection 77
disconnect macro 130
Display logic 36
DOCTYPEs 173
Document Access Definition 78
Document Access Definition Extender 78

E
EAR 256
EJB Web service 21
Enterprise Application Project 256
Index

PHP module 172
PHP SOAP implementations 180
PHP Web services client 242
process choreography 15
ProgramCall 35
ProgramCall bean example 40
programmatic interface 24
Project Explorer 81
Project investments in developing a service 36

Q
QDSPSGNINF 135
Qshell 39

R
Rational Product Updater 197
Recording a connect macro 131
Recording a data macro 140
Recording a disconnect macro 154
Recording macros 130
Redbooks Web site 284
Contact us xi
remote method invocation 249
remote procedure call 19
response message 21
Reviewing the Generated Web service and Web service client code 59
REXX procedure 76
RMI 249
RPC 19
RPG 196
RPG as a Web service Client 196
RPG OPM 76

S
Sample scenario 27
Screen Recognition Criteria 142
Secure Sockets Layer 25
server-side scripting language 173
Service Broker 4
Service Consumer 268
Service Provider 4, 268
Service Requestor 4
Service-Oriented Architecture 4
Business-related example 5
Complex parameter 14
Creating SOA applications from services 15
Data type 14
Identifying business function 12
Implementation 6
Interaction 15
Introduction 4
Loose coupling 7
primitive data types 14
programming language specific data types 14
Quality of services 8
Reuse 7
security considerations 15
Service design considerations 14
service-level agreement 8
Simple parameter 14
String parameter 14
Today's uses of services 13
Setting up the DB2 connection 81
single message authentication 38
Sitepoint 177
skeleton 24
Smarty 177
SMTP 19, 21
SOA characteristics 6
SOA from a business perspective 8
SOA-compliant application design 12
SOAP 19
protocol-independent 19
Web services messaging layer 19
SOAP API 14, 20
SOAP body 19
SOAP client 21
SOAP client proxy 20
SOAP connector port 249
SOAP envelope 19
SOAP Headers 19
SOAP message 18–19
SOAP messaging 38
SOAP monitoring utility 267
SOAP request 20
SOAP server 18
SQL result set 37
SQL statement 79
SQL stored procedure 76
SQL-based access 78
SQL-based operation 79
SQL-based query 79
SSL 21, 25
Starting a Web service application 68
Starting WebSphere Development Studio Client for iSeries 40
stateful Web service 15
stateless Web service 15
stored procedure 76
stub 24
synchronous Web service 15

T
TCPMON 268
TCPMON binary 268
TCPMON uses for security 273
TCPMON uses in debugging services problems 273
Template 176
Testing RPG Web service 48
Testing the JSF Web service client 239
text format 223
TLS 25
Transport Layer Security 25

U
UDDI 24
Building SOA-based Solutions for IBM System i Platform

Implementing service-oriented architecture (SOA) with Web services

Examples of Web services based on ProgramCall, HATS, PHP, and Web Services Client for ILE

Excellent starting point for System i developers on Web services

There is a strong shift in the industry toward reuse of the existing software and hardware resources within the companies to minimize the IT cost. Instead of creating or buying a new solutions, companies are trying to build a set of reusable software components based on the existing applications. These components can be quickly assembled in many different ways to satisfy the business needs of the companies.

This environment is based on service-oriented architecture (SOA) and solutions that support business process automation.

This book provides the detailed information about multiple ways for building SOA-based solutions around the System i platform. The discussion in the book covers the server and client side implementations that include:

- ProgramCall in IBM Toolbox for Java
- Host Access Transformation Services (HATS)
- DB2 Web services
- PHP
- IBM Web Services Client for ILE
- Java-Server Faces (JSF)

Parts of the book are appropriate for CIOs, system architects, and application developers.

For more information:
ibm.com/redbooks

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.