BEA WebLogic
Adapter for File

User Guide

Copyright

Copyright © 2003 BEA Systems, Inc. All Rights Reserved.

Copyright © 2003 iWay Software. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

BEA WebLogic Adapter for File User Guide

Part Number	Date	Release
N/A	March 2003	7.0 with Service Pack 2

Table of Contents

About This Document

The *BEA WebLogic Adapter for File User Guide* is organized as follows:

- Chapter 1, "Introducing the BEA WebLogic Adapter for File," introduces the BEA WebLogic Adapter for File, describes its features, and gives an overview of how it works.

- Chapter 2, "Metadata, Schemas, and Repositories." describes metadata, how to name a schema repository and the schema manifest, how to create a schema, how to store directory and template files for transformations.

- Chapter 3, "Defining an Application View for the BEA WebLogic Adapter for File," describes how metadata is used and how application views are created.

- Chapter 4, "Service and Event Configuration," describes how to add services and events to application views.

- Chapter 5, "BEA WebLogic Adapter for File Integration Using Studio," describes how events are incorporated into workflow design.

- Chapter 6, "Transforming Document Formats," describes how to use Message Format Language files to transform a document.

- Chapter 7, "Using Tracing," describes how to use tracing.

What You Need to Know

This document is written for system integrators who develop client interfaces between file systems and other applications. It describes how to use the BEA WebLogic Adapter for File and how to develop application environments with specific focus on message integration. It is assumed that readers know Web technologies and have a general understanding of Microsoft Windows and UNIX systems.

Related Information

The following documents provide additional information for the associated software components:

- *BEA WebLogic Adapter for File Installation and Configuration Guide*

- *BEA WebLogic Adapter for File Release Notes*

- *BEA Application Explorer Installation Guide*

- BEA WebLogic Server installation and user documentation, which is available at the following URL:

  `http://edocs.bea.com/more_wls.html`

- BEA WebLogic Integration installation and user documentation, which is available at the following URL:

  `http://edocs.bea.com/more_wli.html`

Contact Us!

Your feedback on the BEA WebLogic Adapter for File documentation is important to us. Send us e-mail at docsupport@bea.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the BEA WebLogic Adapter for File documentation.

In your e-mail message, please indicate which version of the BEA WebLogic Adapter for File documentation you are using.

If you have any questions about this version of the BEA WebLogic Adapter for File, or if you have problems using the BEA WebLogic Adapter for File, contact BEA Customer Support through BEA WebSupport at www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number

- Your company name and company address

- Your machine type and authorization codes

- The name and version of the product you are using

- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.

Convention	Item
italics	Indicates emphasis or book titles.

monospace text	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard. *Examples*:
	 `#include <iostream.h> void main ( ) the pointer psz`
	 `chmod u+w *`
	 `\tux\data\ap`
	 `.doc`
	 `tux.doc`
	 `BITMAP`
	 `float`

monospace boldface text	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
monospace italic text	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 SIGNON OR
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

[ ]	Indicates optional items in a syntax line. The brackets themselves should never be typed. *Example*:
	 `buildobjclient [-v] [-o name ] [-f file-list]...`
	 `[-l file-list]...`

Convention	Item
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

...	Indicates one of the following in a command line:
	
	■ That an argument can be repeated several times in a command line
	
	■ That the statement omits additional optional arguments
	
	■ That you can enter additional parameters, values, or other information
	
	The ellipsis itself should never be typed.
	
	Example:
	
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

1   Introducing the BEA WebLogic Adapter for File

This section introduces the BEA WebLogic Adapter for File, describes its features, and gives an overview of how it works. It includes the following topics:

- Introduction

- How the BEA WebLogic Adapter for File Works

Introduction

From the company that delivers the market's fastest growing integration solution comes a standards-based method of implementing the critical "last mile" of connectivity to your enterprise applications. As an extension to BEA WebLogic Integration™, BEA offers a growing portfolio of application, technology, and utility adapters. These best-of-breed adapters completely conform to the J2EE Connector Architecture specification, and feature enhancements that enable faster, simpler, and more robust integration of your business-critical applications.

Since most applications are bound by the file systems of the servers on which they run, or the protocols over which they are communicated, the file systems themselves must be taken into consideration in any enterprise integration strategy. The BEA WebLogic Adapter for File incorporates in-depth knowledge of FTP and file systems to optimize the integration files with enterprise application systems.

The BEA WebLogic Adapter for File enables integration with remote files using File Transport Protocol (FTP). Files within the local file system can also be processed. In both cases, the file contents (or payload) can be XML, non-XML ASCII, or custom data formats. This provides a convenient and simple method for integration with remote application systems using BEA WebLogic Integration.

Key features of the BEA WebLogic Adapter for File include support for:

- Asynchronous, bi-directional message interactions between BEA WebLogic Integration, FTP servers, and the local file system.

- BEA Application Explorer, which makes use of metadata on local and remote file systems to build application views that can be used in the business process management functionality.

- A business process that runs within BEA WebLogic Integration to transfer data to and from FTP servers.

- Integration of service (inbound) and event (outbound) operations in workflows.

- XML, Comma Separated Variable (CSV), Excel, Message Format Language (MFL), and Custom Data Formats. The adapter converts non-XML files into XML formats. Delimited, fixed length, and variable length file formats are supported. Custom Data Formats are expressed using an XML dictionary file, which generates the appropriate schemas required by WebLogic Integration.

- Events that can be routed to general-purpose messaging systems. Message attachments can contain proprietary custom data formats that need to be transformed. The BEA WebLogic Adapter for File supports processing of XML, ASCII (CSV, CDF, Excel) and custom non-XML based messages containing structured, binary, and string data.

How the BEA WebLogic Adapter for File Works

The adapter provides transport protocol support so that it can "listen" and "emit" documents using file system and FTP protocols.

The listening capability has been implemented as an event within WebLogic Integration. When an inbound document is detected, the event provides options that are configured with the BEA Application Explorer and the WebLogic Application View Console:

■ **Transformation services.** XML is quickly becoming the standard for exchanging information between applications; it is invaluable in integrating disparate applications. With this in mind, the BEA WebLogic Adapter for File exposes parse or transformation services. Using pre-built customizable parsers to enable the parsing and conversion of a non-XML formatted document and XSLT transformation to modify XML document format, the BEA WebLogic Adapter for File will ensure that any incoming document can be converted to XML format dictated by your event/service schemas. The BEA WebLogic Adapter for File can also be used in conjunction with other BEA WebLogic Adapters to process a variety of message types, such as SAP IDoc, SWIFT, FIX, HIPAA, and HL7.

The emitting capability has been implemented as a service within WebLogic Integration. When an outbound document is created, the service provides options that are configured with the BEA Application Explorer and WebLogic Application View Console:

■ **Transformation services.** Outbound documents can be parsed or transformed to convert XML documents to non-XML formats. The parsing of non-XML documents can also be provided by an XML file that defines a dictionary for certain file formats or a Java class designed to convert from an outgoing XML document into a non-XML document. This document can then be passed through the standard service transport protocols—File, FTP, SMTP, and HTTP. The BEA WebLogic Adapter for File can also be used in conjunction with other BEA WebLogic Adapters to process a variety of message types, such as SAP IDoc, SWIFT, FIX, HIPAA, and HL7.

- **Error Handling.** It is possible to define an alternative error-to option such that in the event that a remote file server is unavailable the file could be written to a local file system directory for further processing. Or, if the required outbound file system was full, the outbound file could be placed in another directory on a local system or remote FTP server.

2 Metadata, Schemas, and Repositories

This section explains how metadata for your enterprise information system (EIS) is described, how to name a schema repository and the schema manifest, how to create a schema, and how to store directory and template files for transformations. After the metadata for your EIS is described, you can create and deploy application views using the WebLogic Application View Console.

This section includes the following topics:

- Understanding Metadata

- Schemas and Repositories

- The Repository Manifest

- Creating a Schema

- Storing Directory and Template Files for Transformations

Understanding Metadata

When you define an application view, you are creating an XML-based interface between WebLogic Integration and an enterprise information system (EIS) or application within your enterprise. The BEA Adapter for File is used to define a file based interface to applications within and outside of the enterprise. Many applications or information systems use file systems to store and share data. These files contain information required by other applications, and this information can be fed information via the BEA WebLogic Adapter for File.

The BEA WebLogic Adapter for File can read, write, or manipulate different types of files stored in multiple file systems or FTP sites. WebLogic integration uses XML as the common format for data being processed in its workflow, which requires information that is not in XML to be transformed to XML. Alternatively, to share information successfully, the file adapter can transform information from the XML format used in WebLogic Integration to widely used formats, such as commercial XML schemas, EDI, SWIFT, HIPAA, HL7, and others.

For example, Excel is a widely used application that allows all types of professionals (from fund managers to administrative assistants) to collate information pertinent to their working environment. This information can be shared by other applications using the adapter's transformation capability, which can convert a worksheet to XML and to other business partners via an EDI stream.

To map this information within the workflow via event and service adapters, the BEA WebLogic Adapter for File requires XML schemas for identifying and processing these documents. Because some of these documents may be in non-XML form, such as Excel, CSV, SWIFT, or HIPAA, they must be converted to XML and described to WebLogic Integration using these schemas. A manifest file is used to relate schemas to events or services. The schemas and manifest are stored in a folder or directory in the local file system referred to as the EIS repository. The repository location is required when creating an application view from which events and services can be configured.

Events are triggers to workflows. When a particular file arrives at a location, an event can be triggered to read and convert, if necessary, to the XML format that conforms to a particular schema, which then initiates a flow. Services are called from the workflow to perform particular operations, such as file manipulation (move, copy, delete, append and prepend); file reads and file writes and conversion to other formats are also supported.

The adapter converts non-XML, non-self describing documents into XML in two ways. The Format Builder tool can build MFL files that are stored in the WebLogic server local repository. The Format Builder is best used for unconventional or custom format files. The structure of this file can be defined using the Format Builder and used for basic conversion to or from XML. For conventional documents that are not self-describing such as SWIFT, HIPAA, EDI/X12, EDIFACT and HL7, the structure of the data is described using a data dictionary or `.dic` file.

Pre-built dictionaries are supplied for these formats, so creating them is not necessary, but you can customize them to conform with specific electronic trading agreements. Transformation templates or `.xch` files use these dictionaries to map the document to its XML form or vice versa.

Transformation templates use dictionaries as metadata for the file being read or created. The template defines the input value's relationship with the output values using the dictionary and XML schema. For events, the template is used to convert a non-XML format to XML and for services, the conversion can be reversed using an alternative template.

The templates are stored in the templates sub-directory of the EIS repository. Dictionaries are stored in the dictionaries sub-directory. The following is a sample data dictionary.

Listing 2-1   Data Dictionary Sample

```
<?xml version="1.0"?>
<!-- Title = EDI Transaction Dictionary by Transaction Set -->
<!-- Transaction = 276 Health Care Claim Status Request -->

<EDI Type="ASCII" Version="4010" Standard="X12">
<TransactionSet ID="276" Name="Health Care Claim Status Request"
Note="">

<!-- Table 1 -->

   <Segment ID="ST" Name="Transaction Set Header" Req="M"
MaxUse="1">

      <Element ID="01" Name="Transaction Set Identifier Code"
Req="M" Type="ID" MinLength="3" MaxLength="3" Note="The transaction
set identifier 'ST01' is used by the translation routines of the
interchange partners to select the appropriate transaction set
definition 'e.g., 810 select the Invoice Transaction Set'."/>
```

```
     <Element ID="02" Name="Transaction Set Control Number" Req="M"
Type="AN" MinLength="4" MaxLength="9"/>

      <Element ID="03" Name="Implementation Convention Reference"
Req="O" Type="AN" MinLength="1" MaxLength="35" Note="The
implementation convention reference 'ST03' is used by the
translation routines of the interchange partners to select the
appropriate implementation convention to match the transaction set
definition."/>

   <Segment ID="BHT" Name="Beginning of Hierarchical Transaction"
Req="M" MaxUse="1">

      <Element ID="01" Name="Hierarchical Structure Code" Req="M"
Type="ID" MinLength="4" MaxLength="4"/>

      <Element ID="02" Name="Transaction Set Purpose Code" Req="M"
Type="ID" MinLength="2" MaxLength="2"/>

      <Element ID="03" Name="Reference Identification" Req="O"
Type="AN" MinLength="1" MaxLength="50" Note="BHT03 is the number
assigned by the originator to identify the transaction within the
originator's business application system."/>
```

After the metadata for your EIS has been described, application views can be created and deployed using the WebLogic Integration Application View Console. For more information on creating application views, see Chapter 3, "Defining an Application View for the BEA WebLogic Adapter for File."

Schemas and Repositories

You describe all the documents entering and exiting your WebLogic Integration system using W3C XML schemas. These schemas describe each event arriving to and propagating out of an event, and each request sent to and each response received from a service. There is one schema for each event and two for each service (one for the request, one for the response). The schemas are usually stored in files with an .xsd extension.

Use the WebLogic Integration Application View Console to access events and services, and to assign a schema to each event, request, and response. Assign each application view to a schema repository; several application views can be assigned to the same repository.

BEA WebLogic Adapters all make use of a schema repository to store their schema information and present it to the WebLogic Application View Console. The schema repository is a directory containing:

■ A manifest file that describes the event and service schemas.

■ The corresponding schema descriptions.

To work with schemas, you must know how to:

■ Name a schema repository.

■ Create a manifest.

■ Create a schema.

Naming a Schema Repository

The schema repository has a three-part naming convention:

`session_base_directory\adapter\connection_name`

- `session_base_directory` is the schema's session base path, which represents a folder under which multiple sessions of schemas may be held.

- `adapter` is the type of adapter (for example, FILE or SAP).

- `connection_name` is a name representing a particular instance of the adapter type.

For example, if the session base path is `/usr/opt/bea/bse`, the adapter type is FILE, and the connection name is FILEDev, then the schema repository is the directory:

`/usr/opt/bea/bse/FILE/FILEDev`

The Repository Manifest

Each schema repository has a manifest that describes the repository and its schemas. This repository manifest is stored as an XML file named `manifest.xml`.

The following is an example of a sample manifest file showing relationships between events and services and their schemas.

The manifest file relates documents (through their schemas) to services and events. The manifest exposes schema references to the event relating the required document (via the root tag) to the corresponding schema. Schemas and manifests are stored in the same directory, the repository root of the EIS. The following is an example of the a manifest file with a description of the elements.

Listing 2-2   Sample Manifest File

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<manifest>
   <connection/>
   <schemaref name="service_only">
      <request root="INVOICE" file="INVOICE.xsd"/>
      <response root="emitStatus" file="FileEmit.xsd"/>
   </schemaref>
   <schemaref name="event_only">
      <event root="PURCHASE_ORDER" file="PURCHASE_ORDER.xsd"/>
   </schemaref>
   <schemaref name="shared">
      <request root="STOCK_STATUS" file="STOCK_STATUS.xsd"/>
      <response root="emitStatus" file="FileEmit.xsd"/>
      <event root="STOCK_UPDATE" file="STOCK_UPDATE.xsd"/>
   </schemaref>
</manifest>
```

The manifest has a connection section (which is not used by the BEA WebLogic Adapter for File) and a schema reference section, named schemaref. The schema reference name is displayed in the schema drop-down list on the Add Service and Add Event windows in the WebLogic Integration Application View Console. This sample manifest has three schema references or schemaref tags; one for services only, one for events only, and one for a combination of services and events. Events require only one schema, defined by the *event* tag. This relates the root tag of an XML document to a schema in the EIS repository. For services, two schemas are required: one for the document being passed to the service, represented by the *request* tag, and one for the expected *response* document received from the service operation, represented by the *response* tag.

Creating a Repository Manifest

The repository manifest is an XML file with the root element manifest and two sub-elements:

- connection, which appears once, and which you can ignore because it is not used by the BEA WebLogic Adapter for File.

- schemaref, which appears multiple times, once for each schema name, and which contains all three schemas—request, response, and event.

To create a manifest:

1. Create an XML file with the following structure:

```
<manifest>
   <connection>
   </connection>
</manifest>
```

2. For each new event or service schema you define, create a schemaref section using this model:

```
<schemaref name="OrderIn">
   <request root="OrderIn" file="service_OrderIn_request.xsd"/>
   <response root="emitStatus" file="MQEmitStatus.xsd"/>
   <event root="OrderIn" file="event_OrderIn.xsd"/>
</schemaref>
```

Here, the value you assign to:

- file is the name of the file in the schema repository.

- root is the name of the root element in the actual instance documents that will arrive at, or be sent to, the event or service.

Creating a Schema

Schemas describe the rules of the XML documents that will traverse WebLogic Integration. You can generate a schema manually or through a schema-generating tool.

WebLogic Integration interacts with application view events and services by sending and receiving XML messages. The XML messages are defined by XML schemas. The schemas are stored in directories specific for each adapter.

You must set up at least one directory for each adapter you use. This directory can contain multiple subdirectories, each of which can hold schemas specific to different instances of your application. You should name the parent directory to represent your adapter; you can name the subdirectories according to what is appropriate for your application.

For example, if you have four instances of an application that exchanges messages between the BEA WebLogic Adapter for File and WebLogic Integration, you should set up four subdirectories to store the schemas; the subdirectories should be in a parent `File` directory:

```
D:\TraderSystems\BEAapps\File\FTPprod
D:\TraderSystems\BEAapps\File\FTPdev
D:\TraderSystems\BEAapps\File\FTPuat
D:\TraderSystems\BEAapps\File\FILEprod
```

The schemas for the documents being processed are stored within those directories.

The following is an example of an instance document for the OrderIn event referred to in "Creating a Repository Manifest" on page 2-8.

Listing 2-3   Instance Document for OrderIn Event

```xml
<?xml version="1.0"?>

<OrderIn>
   <Store_Code>1003CA</Store_Code>
   <LineItem>
      <Prod_Num>1003</Prod_Num>
      <Quantity>100</Quantity>
      <Price>1.69</Price>
   </LineItem>
   <LineItem>
```

```
                    <Prod_Num>1004</Prod_Num>
                    <Quantity>10</Quantity>
                    <Price>1.79</Price>
            </LineItem>
</OrderIn>
```

The following is a schema matching this instance document and may be manually
coded or generated from any XML editor:

Listing 2-4   Schema Matching OrderIn Event Instance Document

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xsd:element name="OrderIn">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Store_Code"/>
                <xsd:element ref="LineItem" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="LineItem">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Prod_Num"/>
                <xsd:element ref="Quantity"/>
                <xsd:element ref="Price"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Price">
        <xsd:simpleType>
            <xsd:restriction base="xsd:decimal">
                <xsd:enumeration value="1.69"/>
                <xsd:enumeration value="1.79"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="Prod_Num">
        <xsd:simpleType>
            <xsd:restriction base="xsd:short">
                <xsd:enumeration value="1003"/>
```

```
            <xsd:enumeration value="1004"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="Quantity">
    <xsd:simpleType>
        <xsd:restriction base="xsd:byte">
            <xsd:enumeration value="10"/>
            <xsd:enumeration value="100"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="Store_Code" type="xsd:hexBinary"/>
</xsd:schema>
```

Storing Directory and Template Files for Transformations

The BEA WebLogic Adapter for File supports the exchange of XML and non-XML messages with WebLogic Integration. Templates and dictionaries are created and associated with the BEA WebLogic Adapter for File events and services. Dictionaries (.dic extension) are documents that describe an incoming non-XML document. Templates (.xch extension) describe the conversion from one format to another (XML to non-XML, and vice versa). Sample dictionaries and templates are supplied with the product and must be placed in a transform subdirectory in the root directory for your domain, as shown in the following paths:

```
DOMAIN_HOME\transform\xch
DOMAIN_HOME\transform\xslt
DOMAIN_HOME\transform\dic
```

Samples File

Supplied with the BEA WebLogic Adapter for File are sample files (XML and EDI format) that can be used to help test that your environment is correctly set up and working. The BEA_SAMPLES.zip file also includes sample manifest and schema files.

3 Defining an Application View for the BEA WebLogic Adapter for File

This section describes how metadata is used and how application views are created. It includes the following topics:

- Schemas, Dictionaries, and Transformation Templates

- Creating a Transformation Template

- Creating a New Application View

Schemas, Dictionaries, and Transformation Templates

When you define an application view, you are creating an XML-based interface between WebLogic Server and a particular Enterprise Information System (EIS) application within your enterprise. In the case of the BEA WebLogic Adapter for File, this is a set of files that your applications have to create or respond to.

For example, Excel is a widely used application that allows professionals to collate information pertinent to their working environment. SAP is also a valuable application used in the IT environment for CRM solutions. The BEA WebLogic Adapter for File allows you to effectively share information between these disparate systems. The adapter can detect an update in the Excel document, transform it to XML, and pass it to WebLogic Integration. Based on a predefined workflow, WebLogic Integration can use the BEA WebLogic Adapter for SAP to send the information, in the appropriate format, and cause an update in the SAP system. As another example, WebLogic Integration can place the information on a file system, FTP site, or MQSeries queue for integration with applications that comply with industry- or government-mandated standards, such as HL7, HIPAA, or Swift. In this case, WebLogic Integration uses the BEA WebLogic Adapter for File or the BEA WebLogic Adapter for MQSeries in conjunction with the BEA WebLogic Adapter for HL7, the BEA WebLogic Adapter for HIPAA, or the BEA WebLogic Adapter for Swift.

The adapter requires schemas for processing these documents. As some of these documents may be in non-XML form (for example, Excel, CSV, SWIFT, and HIPAA), they will have to be converted to XML and described to WebLogic Integration using schemas. For more information on schemas, see Chapter 2, "Metadata, Schemas, and Repositories."

For non-XML, non-self describing documents, the structure of the data will need to be described using a data dictionary such as the one shown in the following listing:

Listing 3-1   Data Dictionary

```
<?xml version="1.0"?>

<!-- Title = EDI Transaction Dictionary by Transaction Set -->
<!-- Transaction = 276 Health Care Claim Status Request -->

<EDI Type="ASCII" Version="4010" Standard="X12">
<TransactionSet ID="276" Name="Health Care Claim Status Request"
Note="">

<!-- Table 1 -->

   <Segment ID="ST" Name="Transaction Set Header" Req="M"
MaxUse="1">

      <Element ID="01" Name="Transaction Set Identifier Code"
Req="M" Type="ID" MinLength="3" MaxLength="3" Note="The transaction
set identifier 'ST01' is used by the translation routines of the
interchange partners to select the appropriate transaction set
definition 'e.g., 810 select the Invoice Transaction Set'."/>

     <Element ID="02" Name="Transaction Set Control Number" Req="M"
Type="AN" MinLength="4" MaxLength="9"/>

      <Element ID="03" Name="Implementation Convention Reference"
Req="O" Type="AN" MinLength="1" MaxLength="35" Note="The
implementation convention reference 'ST03' is used by the
translation routines of the interchange partners to select the
appropriate implementation convention to match the transaction set
definition."/>

   </Segment>

   <Segment ID="BHT" Name="Beginning of Hierarchical Transaction"
Req="M" MaxUse="1">

      <Element ID="01" Name="Hierarchical Structure Code" Req="M"
Type="ID" MinLength="4" MaxLength="4"/>

      <Element ID="02" Name="Transaction Set Purpose Code" Req="M"
Type="ID" MinLength="2" MaxLength="2"/>

      <Element ID="03" Name="Reference Identification" Req="O"
Type="AN" MinLength="1" MaxLength="50" Note="BHT03 is the number
assigned by the originator to identify the transaction within the
originator's business application system."/>
```

Creating a Transformation Template

You can create a transformation template file by running the Session Connection utility from the BEA Application Explorer. The utility creates the template and the schema automatically and configures the manifest.xml file accordingly. To create a transformation template file, the dictionary must have been defined as described in "Schemas, Dictionaries, and Transformation Templates" on page 3-2.

The templates are stored in the libraries created in the wlidomain directory in the correct folder (transform/xch). Dictionaries need to be stored in the transform/dic directory. For more information on file locations, see the *BEA WebLogic Adapter for File Installation and Configuration Guide.*

Once the metadata for your EIS has been described, application views can be created and deployed using the WebLogic Integration Application View Console.

Creating a New Application View

You can create an application view once the metadata for your EIS has been described.

To create an application view:

1. Open the Application View Console, which is found at the following location:

   http://*host:port*/wlai

   Here, *host* is the TCP/IP address or DNS name where WebLogic Integration Server is installed, and *port* is the socket on which the server is listening. The default port at the time of installation is 7001.

2. If prompted, enter a user name and password, as shown in the following figure.

Figure 3-1   Application View Console Logon Window

> **Note:**   If the user name is not system, it must be included in the adapter group. For more information on adding the administrative server user name to the adapter group, see the *BEA WebLogic Adapter for File Installation and Configuration Guide*.

3. Click Login.

   The WebLogic Integration Application View Console opens.

Figure 3-2   Application View Console Window

4. Click Add Application View to create an application view for the adapter. The Define New Application View dialog box opens. An application view enables a set of business processes for this adapter's target EIS application. For more information, see "Defining an Application View" in *Using Application Integration*:

   - For WebLogic Integration 7.0, see
     `http://edocs.bea.com/wli/docs70/aiuser/2usrdef.htm`

   - For WebLogic Integration 2.1, see
     `http://edocs.bea.com/wlintegration/v2_1sp/aiuser/2usrdef.htm`

Figure 3-3 Define New Application View Window

5. In the Application View Name field, enter a name. The name should describe the set of functions performed by this application. Each application view name must be unique to its adapter. Valid characters are a-z, A-Z, 0-9, and _ (underscore).

6. In the Description field, enter any relevant notes. These notes are viewed by users when they use this application view in workflows using business process management functionality.

7. From the Associated Adapter list, select BEA_FILE_1_0 to associate the BEA WebLogic Adapter for File with this application view.

8. Click OK. The Configure Connection Parameters window opens.

Figure 3-4   Configure Connection Parameters Window
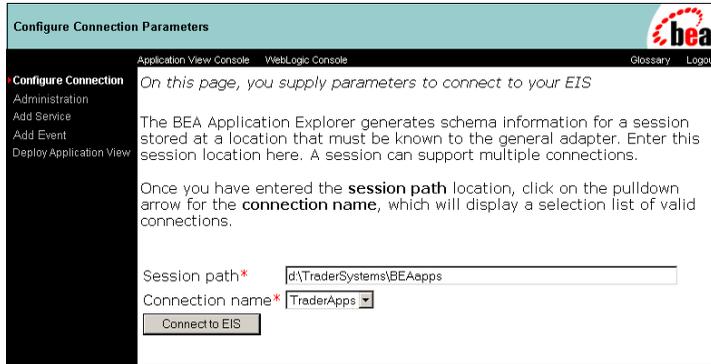
9. Enter the root directory containing your schema subdirectories. For example,
    `d:\TraderSystems\BEAapps.`

10. Select the connection name (the subdirectory containing schemas and the
    manifest file) from the drop-down list. For example, `TraderApps.`

    For more information on schemas and the manifest file, see Chapter 2,
    "Metadata, Schemas, and Repositories."

 Click Connect to EIS to view the Application View Console Administration window.

Figure 3-5   Application View Console Administration Window
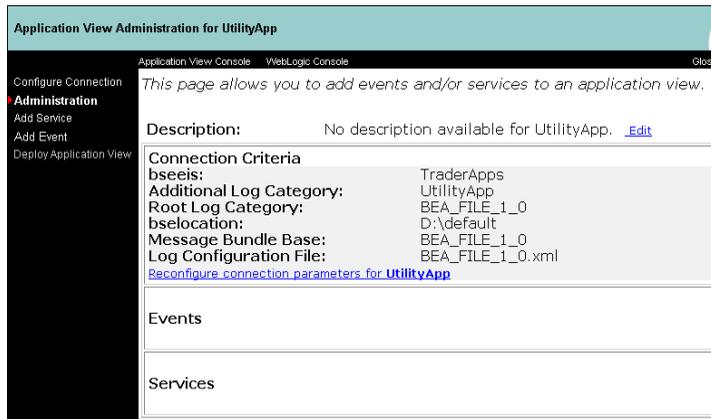
You can now configure services and events as described in Chapter 4, "Service and
Event Configuration."

4 Service and Event Configuration

This section describes how to add services and events to application views. It contains the following topics:

- Adding a Service to an Application View
- Adding an Event to an Application View
- Deploying an Application View
- Testing a Service or Event

Adding a Service to an Application View

After you create and configure an application view as described in Chapter 3, "Defining an Application View for the BEA WebLogic Adapter for File," you can add services that support the application's functions.

To add a service to an application view:

1. If it is not already open, open the application view to be modified. For more information, see "Editing an Application View" in "Defining an Application View" in *Using Application Integration*:

   - For WebLogic Integration 7.0, see
     `http://edocs.bea.com/wli/docs70/aiuser/2usrdef.htm`

- For WebLogic Integration 2.1, see
  `http://edocs.bea.com/wlintegration/v2_1sp/aiuser/2usrdef.htm`

2. If the application view is deployed, you must undeploy it before adding the service. See "Optional Step: Undeploying an Application View" in "Defining an Application View" at the URL referenced in the previous step.

3. In the left pane, click Administration from the Configure Connection list. The Application View Console Administration window opens.

Figure 4-1   Application View Console Administration Window
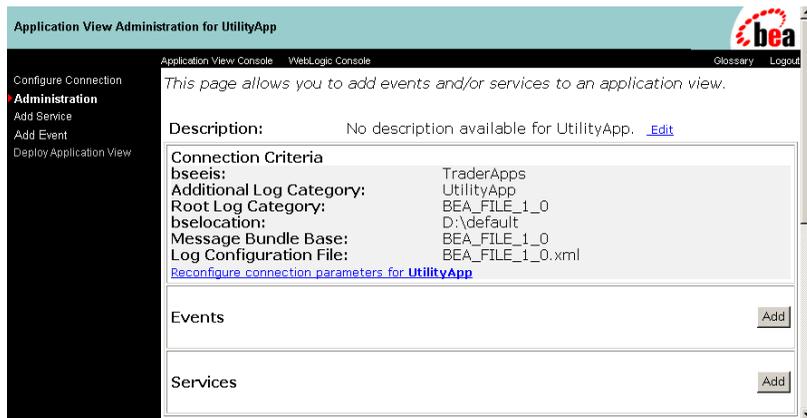
4. Click Add in the Services pane.

   The Add Service window opens.

Figure 4-2   Add Service Window

5. In the Unique Service Name field, enter a name. The name should describe the function performed by this service. Each service name must be unique to its application view. Valid characters are a-z, A-Z, 0-9, and _ (underscore).

6. Select the operation to be configured from the Select drop-down list. This list includes: File System Write, File System Read, FTP Write, and File Operations. You can configure only one operation per service.

7. Select the protocol or format from the Transform engine drop-down list, and enter the required values (required fields are marked with an asterisk). Descriptions of the parameters for each configured operation are provided in the following tables:

Table 4-1 File System Write

Setting	Meaning/Properties
Character Set Encoding* (*Required)	**Type/Value:** String **Description:** The character encoding system to use for the document being emitted. It defaults to UTF-8.

Table 4-1 File System Write (Continued)

Setting	Meaning/Properties
Transform name	**Type/Value:** String **Description:** Parser name and settings required for transformation. **Values:** Name and extension of the transformation template. If it is not stored in the standard location, enter the full path. **Note:** Do not enter an `.mfl` extension for a Message Format Language (MFL) file; these files are not stored with an extension.
type	**Type/Value:** Drop-down list **Description:** Format of the document being written. **Values:** XML format or flat format for non-XML output.
Transform engine	**Type/Value:** Drop-down list **Description:** The engine required to transform the document. Select Message Format Language (`mfl`), XSL Transformation (`xslt`), or Supplied transformation templates (`xch`).
directory* (*Required)	**Type/Value:** Directory Path **Description:** Directory to which output messages are emitted.
output file name/mask* (*Required)	**Type/Value:** String **Description:** The output file name (can contain a '*'), which gets expanded to a timestamp. A pound symbol can be used as a mask for a sequence count. Each pound symbol represents a whole number integer value. For example, File## counts up to 99 before restarting at 0, File### counts up to 999 before restarting at 0, and so on.
Error Local Directory	**Type/Value:** Directory Path **Description:** Directory to which the document being processed is written to in the event of a failure (for instance if the transformation fails or the document does not match the schema).
Trace on/off	**Type/Value:** Check box Description: Generates a basic trace that displays the input XML (up to 300 bytes) before parsing, and shows the request being processed. For more information about tracing, see Chapter 7, "Using Tracing."
Verbose Trace on/off	**Type/Value:** Check box **Description:** Generates a trace that displays configuration parameters used by the adapter. For more information about tracing, see Chapter 7, "Using Tracing."

Table 4-1 File System Write (Continued)

Setting	Meaning/Properties
Document Trace on/off	**Type/Value:** Check box **Description:** Generates a trace that displays the input document after it was analyzed and the response document being returned. For more information about tracing, see Chapter 7, "Using Tracing."

Table 4-2 File System Read

Setting	Meaning/Properties
Character Set Encoding* (*Required)	**Type/Value:** String **Description:** The character encoding system used in the document being read. It defaults to UTF-8.
Input tag containing filename* (*Required)	**Type/Value:** String **Description:** Name of the XML tag that contains the name of the file in the file system. The file can be XML or non-XML format and can be transformed as part of the configuration.
Tag to enclose data read	**Type/Value:** String **Description:** If the non-XML document is not being transformed and must be embedded in an XML tag, supply the tag name here.
Parent directory (if filename is not absolute)	**Type/Value:** Directory Path **Description:** Optional path to the file, if it is not part of the Input tag.
Format	**Type/Value:** Drop-down list **Description:** Format of the file being read **Value:** xml - XML format input **Value:** flat - non-XML input format
Transform name	**Type/Value:** String **Description:** Parser name and settings required for transformation. **Values:** Name and extension of the transformation template. If it is not stored in the standard location, enter the full path. **Note:** Do not enter an `.mfl` extension for a Message Format Language (MFL) file; these files are not stored with an extension.

Table 4-2 File System Read (Continued)

Setting	Meaning/Properties
Transform engine	**Type/Value:** Drop-down list **Description:** The engine required to transform the document. Select Message Format Language (`mfl`), XSL Transformation (`xslt`), or Supplied transformation templates (`xch`).
Error Local Directory	**Type/Value:** Directory Path **Description:** Directory to which the document being processed is written to in the event of a failure (for instance if the transformation fails or the document does not match the schema).
Trace on/off	**Type/Value:** Check box **Description:** Generates a basic trace that displays the input XML (up to 300 bytes) before parsing, and shows the request being processed. For more information about tracing, see Chapter 7, "Using Tracing."
Verbose Trace on/off	**Type/Value:** Check box **Description:** Generates a trace that displays configuration parameters used by the adapter. For more information about tracing, see Chapter 7, "Using Tracing."
Document Trace on/off	**Type/Value:** Check box **Description:** Generates a trace that displays the input document after it was analyzed and the response document being returned. For more information about tracing, see Chapter 7, "Using Tracing."

Table 4-3 FTP Write

Setting	Meaning/Properties
Character Set Encoding* (*Required)	**Type/Value:** String **Description:** The character encoding system used in the document being read. It defaults to UTF-8.

Table 4-3 FTP Write (Continued)

Setting	Meaning/Properties
Transform name	**Type/Value:** String **Description:** Parser name and settings required for transformation. **Values:** Name and extension of the transformation template. If it is not stored in the standard location, enter the full path. **Note:** Do not enter an `.mfl` extension for a Message Format Language (MFL) file; these files are not stored with an extension.
type	**Type/Value:** Drop-down list **Description:** Format of the file being read. **Values:** XML format or flat format for non-XML output.
Transform engine	**Type/Value:** Drop-down list **Description:** The engine required to transform the document. Select Message Format Language (`mfl`), XSL Transformation (`xslt`), Supplied transformation templates (`xch`).
Host name* (*Required)	**Type/Value:** String **Description:** FTP target system.
Port number	**Type/Value:** Numeric **Description:** FTP target system port (leave empty for FTP default).
User ID* (*Required)	**Type/Value:** String **Description:** User account ID to use when connecting to the protocol host.
Password* (*Required)	**Type/Value:** String **Description:** Password for the user account to use when connecting to the protocol host.
destination* (*Required)	**Type/Value:** String **Description:** Directory to address on the FTP target system.
output file name/mask*	**Type/Value:** String **Description:** The output file name (can contain a '*'), which gets expanded to a timestamp.

Table 4-3 FTP Write (Continued)

Setting	Meaning/Properties
Retry Interval	**Type/Value:** Retry interval duration in *xx*H:*xx*M:*xx*S format. (for example, 1H:2M:3S, which is 1 hour 2 minutes and 3 seconds) **Description:** The maximum wait interval between retries when a connection fails.
Maxtries	**Type/Value:** String **Description:** Number of retries for a failed attempt to write.
Error Local Directory	**Type/Value:** Directory Path **Description:** Directory to which the document being processed is written to in the event of a failure (for instance if the transformation fails or the document does not match the schema).
Trace on/off	**Type/Value:** Check box **Description:** Generates a basic trace that displays the input XML (up to 300 bytes) before parsing, and shows the request being processed. For more information about tracing, see Chapter 7, "Using Tracing."
Verbose Trace on/off	**Type/Value:** Check box **Description:** Generates a trace that displays configuration parameters used by the adapter. For more information about tracing, see Chapter 7, "Using Tracing."
Document Trace on/off	**Type/Value:** Check box **Description:** Generates a trace that displays the input document after it was analyzed and the response document being returned. For more information about tracing, see Chapter 7, "Using Tracing."

Table 4-4 FTP System Read

Setting	Meaning/Properties
Character Set Encoding* (*Required)	**Type/Value:** String **Description:** The character encoding system used in the document being read. It defaults to UTF-8.

Table 4-4 FTP System Read (Continued)

Setting	Meaning/Properties
Input tag containing filename*	**Type/Value:** String (*Required) **Description:** Name of the XML tag that contains the name of the file in the file system. The file can be XML or non-XML format and can be transformed as part of the configuration.
tag to enclose data read	**Type/Value:** String **Description:** If the non-XML document is not being transformed and needs to be embedded in an XML tag, supply tag name here
Parent directory (if filename is not absolute)	**Type/Value:** Directory **Description:** Optional path to the file, if not part of Input tag.
format	**Type/Value:** Dropdown list **Description:** Format of file being read. **Value:** xml - XML format input **Value:** flat - no-xml input format.
Host name* *Required)	**Type/Value:** String **Description:** FTP target system.
Port number	**Type/Value:** Numeric **Description:** FTP target system port (leave empty for FTP default).
User ID* (*Required)	**Type/Value:** String **Description:** User account ID to use when connecting to protocol host.
Password* (*Required)	**Type/Value:** String **Description:** Password for user account to use when connecting to protocol host.
Transform name	**Type/Value:** String **Description:** Parser name and settings required for transformation. Values: Name and extension of the transformation template. If it is not stored in the standard location, supply the full path.

Table 4-4 FTP System Read (Continued)

Setting	Meaning/Properties
Transform engine	**Type/Value:** Drop-down list **Description:** The engine required to transform the document (Message Format Language (`mfl`), XSL Transformation (`xslt`), Supplied transformation templates (`xch`).
Error Local directory	**Type/Value:** Directory Path. **Description:** Directory to which document being processed is written to in the event of a failure (for instance if the transformation fails or document does not match schema).
Trace on/off	**Type/Value:** Check box **Description:** Generates a basic trace that displays the input XML (up to 300 bytes) before parsing, and shows the request being processed. For more information about tracing, see Chapter 7, "Using Tracing."
Verbose Trace on/off	**Type/Value:** Check box **Description:** Generates a trace that displays configuration parameters used by the adapter. For more information about tracing, see Chapter 7, "Using Tracing."
Document Trace on/off	**Type/Value:** Check box **Description:** Generates a trace that displays the input document after it was analyzed and the response document being returned. For more information about tracing, see Chapter 7, "Using Tracing."

Table 4-5 File Operations

Setting	Meaning/Properties
Relative path setting on/off	**Type:** Check box **Value:** True if only relative path operations are permitted, false if absolute path operations are permitted.
Source Directory	Path base for <from> or <file>
Target Directory	Path base for <to>

Table 4-5 File Operations

Setting	Meaning/Properties
Error Local Directory	**Type/Value:** Directory Path **Description:** Directory to which the document being processed is written to in the event of a failure (for instance if the transformation fails or the document does not match the schema).
Trace on/off	**Type/Value:** Check box **Description:** Generates a basic trace that displays the input XML (up to 300 bytes) before parsing, and shows the request being processed. For more information about tracing, see Chapter 7, "How Adapters Use Tracing."
Verbose Trace on/off	**Type/Value:** Check box **Description:** Generates a trace that displays configuration parameters used by the adapter. For more information about tracing, see Chapter 7, "How Adapters Use Tracing."
Document Trace on/off	**Type/Value:** Check box **Description:** Generates a trace that displays the input document after it was analyzed and the response document being returned. For more information about tracing, see Chapter 7, "How Adapters Use Tracing."
root to transform template directory	**Type/Value:** string **Description:** Location of the xsd files used for transformations.
root to XML style sheet directory	**Type/Value:** string **Description:** Location of the style sheet file (.xsl) used for the xslt transformations.

The File Operations service supports file operations on a local computer. The input document describes one or more operations, and the results of those operations are inserted into the document as a "receipt."

```
<fileop>
      <op>rename | copy | move | delete | append
          <from>name</from>
          <to>name</to>
          <file>name</file>
      </op>
</fileop>
```

The <op> tag is repeatable.

Delete uses the <file> tag.

Copy, move, rename, and append use the <to> and <from> tags.

For example,

```
<fileop>
<op>copy
<from>d:\fileout\mfl\sprokout1.mfl</from>
<to>d:\filein\mfl\sprokout#.mfl</to>
</op>
</fileop>
```

The File Operation will return an answer set with the status of the operation in the format shown below:

```
<fileop>0
      <op>rename | copy | move | delete
          <from>name</from>
          <to>name</to>
          <file>name</file>
          <status>code</status>
          <msg>text</msg>
      </op>
</fileop>
```

The following is an example of the answer set:

```
<fileop>
      <op>copy
          <from>d:\fileout\mfl\sprokout1.mfl</from>
          <to>d:\filein\mfl\sprokout#.mfl</to>
          <status>0</status>
          <msg>success</msg>
      </op>
</fileop>
```

If relative is true, all file operations are relative to the Source Directory and Target Directory directories. In this case, these parameters are mandatory. For security reasons, a customer will probably require relative operations.

The following figure shows the configuration of a File System Write operation:

Figure 4-3   Add Service Window
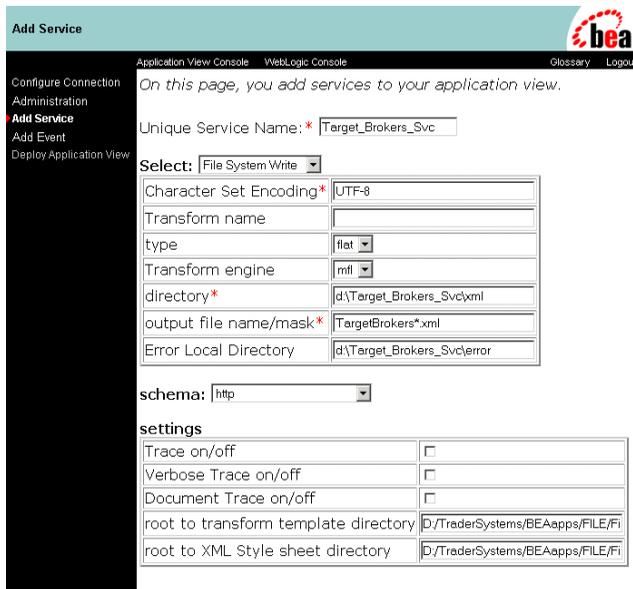
8.   Select the schema (at the bottom of the window) required for this service.

9.   If required, update the settings for the location of the transform or stylesheet.

10.   Click Add.

Figure 4-4   Application View Administration Window
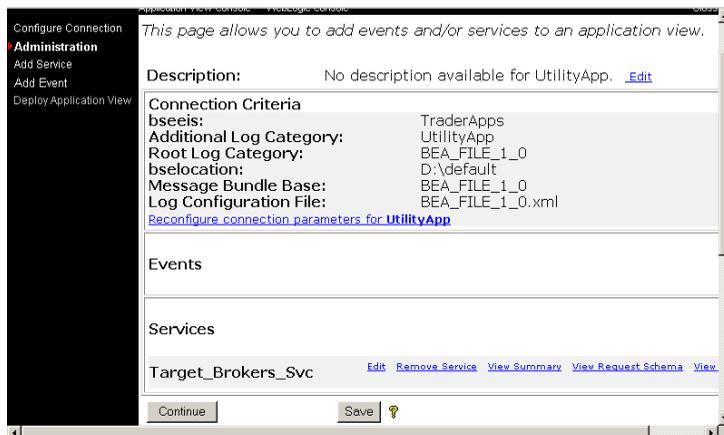
At this point, the application can be deployed or more services or events can be configured. Once the application has been deployed, you can test the service. To deploy the application, see "Deploying an Application View" on page 4-22. To test the application, see "Testing a Service or Event" on page 4-24.

Adding an Event to an Application View

To add events to the application view, schemas must be present and mapped to the BEA WebLogic Adapter for File EIS that is configured for the application view. For more information on creating an application view, see Chapter 3, "Defining an Application View for the BEA WebLogic Adapter for File."

1. If your application is deployed, you will need to undeploy the application and then edit the application view.

Figure 4-5   Application View Console Administration Window
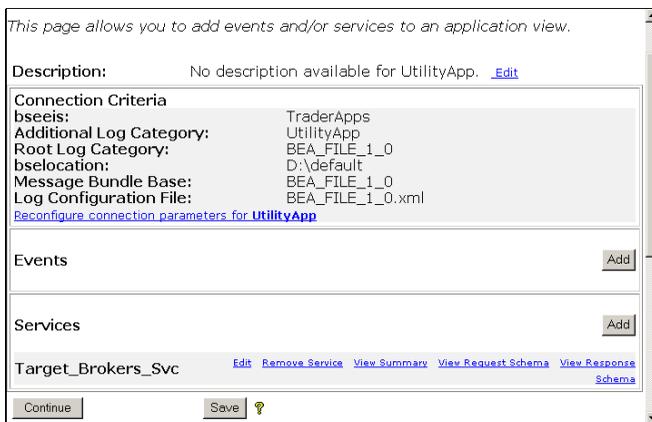
2. From the Application View Console Administration window, click Add in the Events section of the administration pane.

The Add Event window opens.

Figure 4-6   Add Event Window

3. Enter a Unique Event Name.

4. Select the event protocol required from the Select drop-down box.

   You have the option to configure an event based on one of two protocols (File System or FTP).

5. Enter the required values (required fields are marked with an asterisk). Descriptions of the parameters are provided in the following tables:

Table 4-6 File System

Setting	Meaning/Properties
Location* (*Required)	**Type/Value:** Directory Path **Description:** Directory in which input messages are received. The listener allows DOS-style file patterns for input selection. For example, the user can enter the pattern `c:\xyz\AB*CD` to select ABxxxCD, or `c:\xyz\AB?CD` to select ABxCD. **Note:** Do not enter the file extension. This is entered in the File Suffix field. If a pattern is used, the files are selected based in order of suffix and then pattern.
File Suffix* (*Required)	**Type/Value:** String **Description:** Limits input files to those with the specified extensions (separated by commas). For example, `xml,in` Do not use periods (`.`) as they indicate no extension. **Note:** If the file suffix is zip, the unzipped files need to conform to the event schema or they will fail. This function also works with transform configured.
Character Set Encoding* (*Required)	**Type/Value:** String **Description:** The character encoding system to use. It defaults to UTF-8.
Polling interval	**Type/Value:** Polling interval duration in $xxH:xxM:xxS$ format. (for example, 1H:2M:3S, which is 1 hour 2 minutes and 3 seconds) **Description:** The maximum wait interval between checks for new documents. The higher this value, the longer the interval, and the fewer system resources that are used. The side effect of a high value is that the worker thread will not be able to respond to a stop command. If timeout is set to 0, the listener will run once and terminate. Default is 2 seconds.
Sort	**Type/Value:** Boolean (true/false) **Description:** Sort by arrival - If set, sort incoming documents by arrival time. Maintains sequence, but slows performance.
Scan sub-directories	**Type/Value:** Boolean (true/false) **Description:** When checked, scans all subdirectories for documents to be processed.
File-read limit (per scan)	**Type/Value:** Integer **Description:** The number of files read per sweep of the File directory location.

Table 4-6 File System (Continued)

| Setting	Meaning/Properties

transform	**Type/Value:** String (parameters)
	Description: This is where the specific preparse/transform information is specified.
	Values:
	   - entag - this is a preparser that brackets a document with an XML element, enabling you to import a non-XML document into an XML system. You specify the element name in entag's parameter.
	
	For example, `entag(`*`tagname`*`)` produces an XML document of the form
	`<`*`tagname`*`>`
	`John Smith,123 Main Street,NY,NY,10121,$100.00`
	`</`*`tagname`*`>` - excel – used for parsing and transforming Excel documents. Two parameters can be passed to this parser. HAS_HEADERS uses the row 1 headings as tag names for the XML document. NO_HEADERS assumes that the whole spreadsheet is data and defines tag names based on column number (col1, col2, col3, and so on). - mfl – this is the preparser that uses Message Format Language based transformations created using the WebLogic Integration Format Builder, for example, `mfl(`*`mflname`*`)`. - transform – this is the general preparser that uses `.xch` files and dictionaries (`.dic`), where the data is non-self describing. The value that needs to be passed to the transform is the `.xch` file name (with extension), for example, `transform(CSVtoXML.xch)`.

xslt transform	**Type/Value:** String **Description:** The name of the xslt file to be used to transform the incoming XML document.
Copy to Directory	**Type/Value:** Directory path **Description:** The directory to which an incoming document will be copied. Use this feature to keep a copy of the original document.
Error Local Directory	**Type/Value:** Directory Path **Description:** Directory to which the document being processed is written to in the event of a failure (for instance if the transformation fails or the document does not match the schema).

Table 4-6 File System (Continued)

Setting	Meaning/Properties
Trace on/off	**Type/Value:** Check box
	Description: Generates a basic trace that displays the input XML (up to 300 bytes) before parsing, and shows the request being processed. For more information about tracing, see Chapter 7, "Using Tracing."
Verbose Trace on/off	**Type/Value:** Check box
	Description: Generates a trace that displays configuration parameters used by the adapter. For more information about tracing, see Chapter 7, "Using Tracing."
Document Trace on/off	**Type/Value:** Check box
	Description: Generates a trace that displays the input document after it was analyzed and the response document being returned. For more information about tracing, see Chapter 7, "Using Tracing."
root to transform template directory	**Type/Value:** string
	Description: Location of the xsd files used for transformations.
root to XML style sheet directory	**Type/Value:** string
	Description: Location of the style sheet file (.xsl) used for the xslt transformations.

Table 4-7 FTP

Setting	Meaning/Properties
User ID* (*Required)	**Type/Value:** String
	Description: User account to use when connecting to the protocol host.
Password* (*Required)	**Type/Value:** String
	Description: Password for the user account to use when connecting to the protocol host.
Host name* (*Required)	**Type/Value:** String
	Description: Name of host the machine where the listener will contact the service to obtain requests.

Table 4-7 FTP (Continued)

Setting	Meaning/Properties
Location* (*Required)	**Type/Value:** Directory path and file **Description:** Location on FTP host to retrieve files from.You must append the file suffix (extension) to the file or files specified in the Location field. For example, you can enter a specific file such as `/path/to/my/ftp/directory/myfile.xml` or a group of files such as `/path/to/my/ftp/directory/*.zip`.
File suffix	**Type/Value:** String **Description:** The suffix for the filename.
Character Set Encoding* (*Required)	**Type/Value:** String **Description:** The character encoding system to use. It defaults to UTF-8.
Polling interval	**Type/Value:** Polling interval duration in xxH:xxM:xxS format. (for example, 1H:2M:3S, which is 1 hour 2 minutes and 3 seconds) **Description:** The maximum wait interval between checks for new documents. The higher this value, the longer the interval, and the fewer system resources that are used. The side effect of a high value is that the worker thread will not be able to respond to a stop command. If timeout is set to 0, the listener will run once and terminate. Default is 2 seconds.

Table 4-7 FTP (Continued)

Setting	Meaning/Properties
transform	**Type/Value:** String (parameters)
	Description: This is where the specific preparse/transform information is specified.
	Values:
	■ entag - this is a preparser that brackets a document with an XML element, enabling you to import a non-XML document into an XML system. You specify the element name in entag's parameter.
	For example, entag(*tagname*) produces an XML document of the form
	``` <tagname> John Smith,123 Main Street,NY,NY,10121,$100.00 </tagname> ```
	■ excel – used for parsing and transforming Excel documents. Two parameters can be passed to this parser. HAS_HEADERS uses the row 1 headings as tag names for the XML document. NO_HEADERS assumes that the whole spreadsheet is data and defines tag names based on column number (col1, col2, col3, and so on).
	■ mfl – this is the preparser that uses Message Format Language based transformations created using the WebLogic Integration Format Builder, for example, mfl(*mflname*).
	■ transform – this is the general preparser that uses .xch files and dictionaries (.dic), where the data is non-self describing. The value that needs to be passed to the transform is the .xch file name (with extension), for example, transform(CSVtoXML.xch).
xslt transform	**Type/Value:** String
	Description: The name of the xslt file to be used to transform the incoming XML document.
Error Local Directory	**Type/Value:** Directory Path
	Description: Directory to which the document being processed is written to in the event of a failure (for instance if the transformation fails or the document does not match the schema).
Trace on/off	**Type/Value:** Check box
	Description: Generates a basic trace that displays the input XML (up to 300 bytes) before parsing, and shows the request being processed. For more information about tracing, see Chapter 7, "Using Tracing."

Table 4-7 FTP (Continued)

Setting	Meaning/Properties
Verbose Trace on/off	**Type/Value:** Check box **Description:** Generates a trace that displays configuration parameters used by the adapter. For more information about tracing, see Chapter 7, "Using Tracing."
Document Trace on/off	**Type/Value:** Check box **Description:** Generates a trace that displays the input document after it was analyzed and the response document being returned. For more information about tracing, see Chapter 7, "Using Tracing."

6. Select the schema (at the bottom of the window) required for this event.

7. If required, the settings options for the location of the transform or stylesheet can be changed from the default location.

8. Click Add. After adding the event process, the application view administration window opens.

Figure 4-7   Application View Administration Window
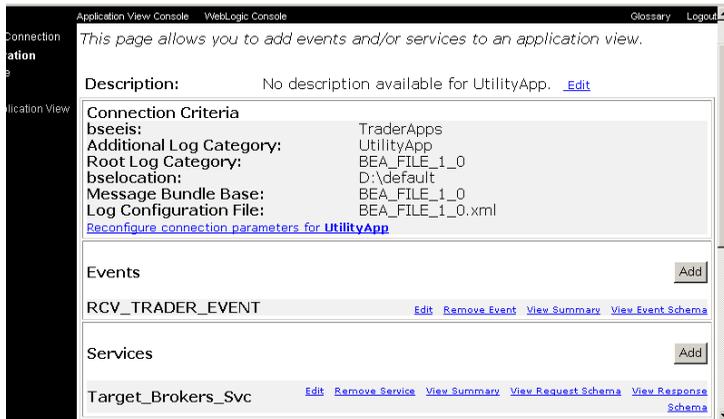
9. Click Save to save your settings.

Deploy your application view (complete with configured events and/or services) by following the steps described in "Deploying an Application View" on page 4-22. Then test your application view by following the steps described in "Testing a Service or Event" on page 4-24.

Deploying an Application View

You can deploy an application view when you have added at least one event or service to it. You must deploy an application view before you can test its services and events or use the application view in the WebLogic Server environment. Application view deployment places relevant metadata about its services and events into a run-time metadata repository. Deployment makes the application view available to other WebLogic Server clients. This means business processes can interact with the application view, and you can test the application view's services and events.

After you configure an event or service, you can deploy your application view from the Application View Console Administration window.

1. If it is not already open, open the application view to be deployed. For more information, see "Editing an Application View" in "Defining an Application View" in *Using Application Integration*:

   - For WebLogic Integration 7.0, see
     `http://edocs.bea.com/wli/docs70/aiuser/2usrdef.htm`

   - For WebLogic Integration 2.1, see
     `http://edocs.bea.com/wlintegration/v2_1sp/aiuser/2usrdef.htm`

Figure 4-8   Edit View Window
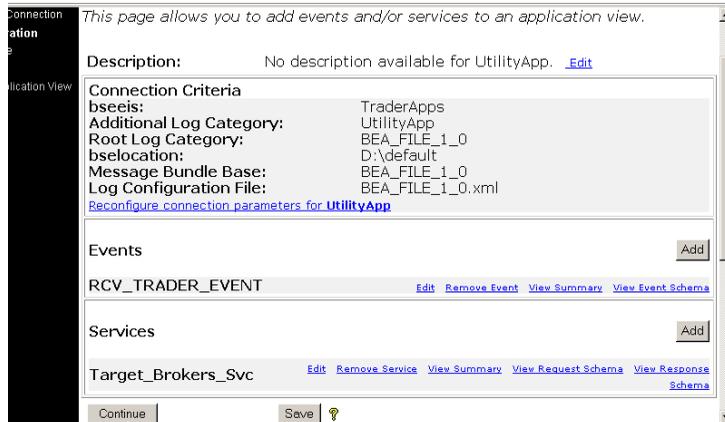
2. From the Edit View window, click Continue. The Deploy Application View window opens.

Figure 4-9   Deploy Application View Window
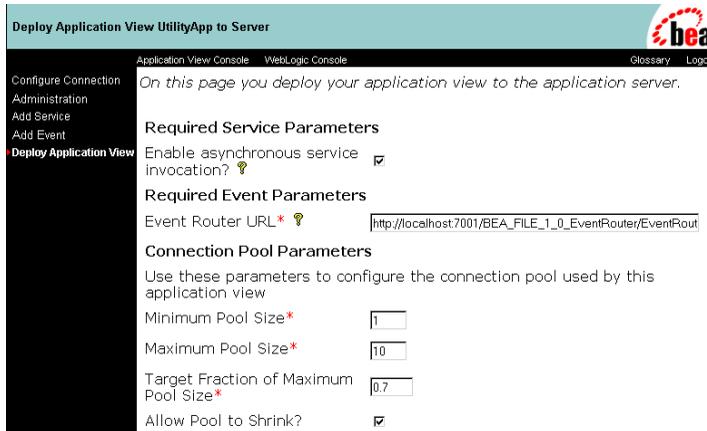
Note: To enable other authorized clients to asynchronously call the services (if any) of this application view, select Enable asynchronous service invocation.

3. To deploy the application view, click Deploy Application View. The Summary for Application View window opens.

    Note: You may decide to click Save and deploy the application view later.

Figure 4-10   Summary for Application View Window

After you create and deploy an application view, you can test the service and events. For more information, see "Testing a Service or Event" on page 4-24.

Testing a Service or Event

After you create and deploy an application view, you can test the services and events.

1. In the Summary for Application View window, click Test for the configured service or event. The Test Service window opens.

Figure 4-11   Test Service Window
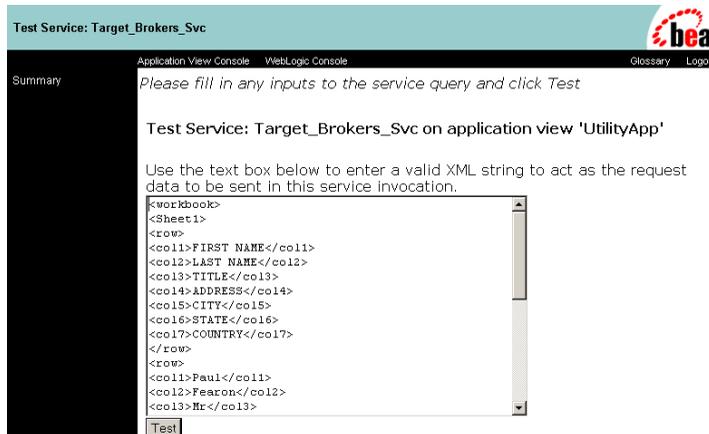
2. Enter the required XML by cutting and pasting a sample XML document. You can use the sample `Headers.xml` supplied with the product.

3. Click Test. If your service has been configured correctly, you receive a response from the file emit process with a status code of "0." Also, you will find that the file has been written to the correct location.

Figure 4-12   Test Service Window

Once you have confirmed that the file has been written correctly (in the correct format, if transformation has been configured) your service or event has been successfully configured.

You can now write custom code to exploit the adapter or create a process flow in Studio. For more information, see "Using Application Views in the Studio" in *Using Application Integration*:

- For WebLogic Integration 7.0, see
  `http://edocs.bea.com/wli/docs70/aiuser/3usruse.htm`

- For WebLogic Integration 2.1, see
  `http://edocs.bea.com/wlintegration/v2_1sp/aiuser/3usruse.htm`

5 BEA WebLogic Adapter for File Integration Using Studio

This section describes how events are incorporated into workflow design. It includes the following topic:

■ Business Process Management Functionality

Business Process Management Functionality

You can integrate your application view, including services and events, into WebLogic Integration business process management workflow tasks.

The following window depicts a simple workflow design triggered by an event described in Chapter 4, "Service and Event Configuration." The incoming event converts a document and utilizes a service to propagate the event to a file system (for example, an Excel file taken from an FTP directory is converted to XML and placed in a local file system). The event will respond to an Excel-formatted document being placed in a file system, then convert the Excel document into XML format, and propagate it (using the workflow) to the service.

Figure 5-1   Workflow Design Window
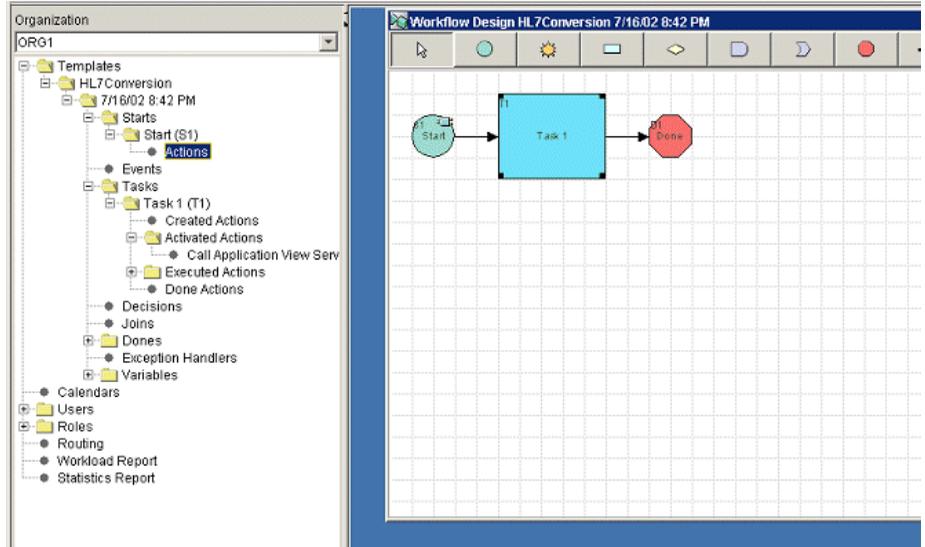
The following window depicts the example Excel document being used in this process. This Excel file is contained in a zip file called BEA_FILE_INSTALL.ZIP, which is supplied with your installation package.

Figure 5-2   Original Excel Document

Starting with the original document, this process essentially consists of the following steps:

1. The document is placed onto an FTP directory, which had a BEA WebLogic Adapter for File event configured to be triggered by the arrival of the Excel document. The event has been pre-configured to apply an Excel to XML conversion (using the preparse parameter). Once the adapter completes the process, the workflow then propagates the XML document onto the adapter service

(configured to route the XML document to a local file system). The service can easily be configured to convert the XML passed to it (by the event router) to another supported format (for example, CSV, HL7, and HIPAA).

2. Once the service has completed its task, the emission report is returned to the workflow.

Figure 5-3   Emission Report
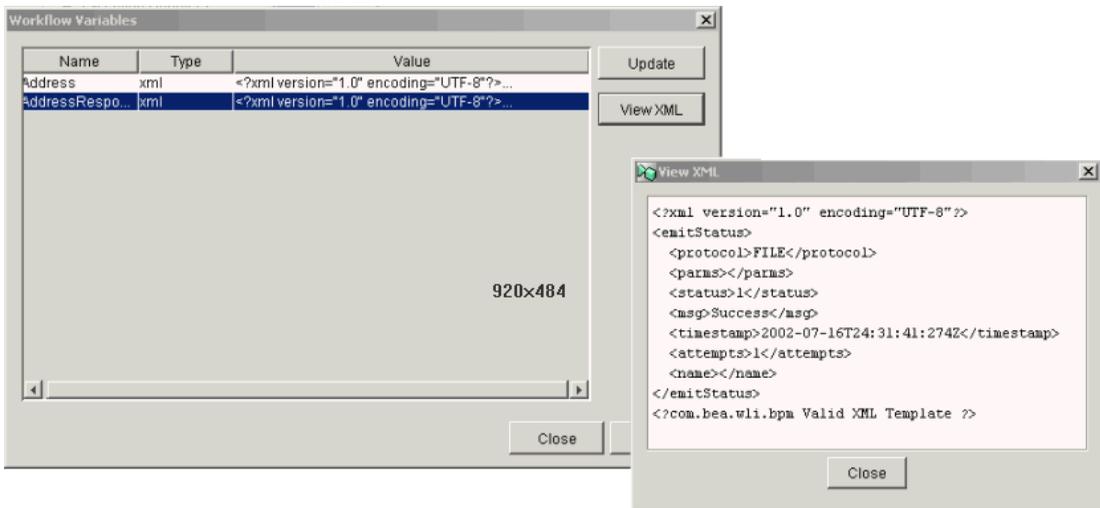
3. The resulting XML file can then be used with back-end systems that may be expecting an XML document in the file system configured by the adapter service.

Figure 5-4   XML File

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<workbook>
  <Sheet1>
    <row>
      <col1>FIRST NAME</col1>
      <col2>LAST NAME</col2>
      <col3>TITLE</col3>
      <col4>ADDRESS</col4>
      <col5>CITY</col5>
      <col6>STATE</col6>
      <col7>COUNTRY</col7>
    </row>
    <row>
      <col1>Paul</col1>
      <col2>Fearon</col2>
      <col3>Mr</col3>
      <col4>2 Penn Plaza</col4>
      <col5>New York</col5>
      <col6>New York</col6>
      <col7>USA</col7>
    </row>
    <row>
      <col1>Jeff</col1>
      <col2>Paoletti</col2>
      <col3>Sir</col3>
      <col4>2 Penn Plaza</col4>
      <col5>New York</col5>
      <col6>New York</col6>
      <col7>USA</col7>
    </row>
    <row>
      <col1>Jonathon</col1>
      <col2>Platt</col2>
      <col3>Lord</col3>
      <col4>Wembley Point, Harrow Road</col4>
      <col5>London</col5>
      <col6>MIDDX</col6>
      <col7>UK</col7>
    </row>
  </row>
```

6 Transforming Document Formats

This section describes how to utilize Message Format Language (MFL) files to transform a document. It includes the following topic:

■ Message Format Language Transformations

Message Format Language Transformations

The BEA WebLogic Adapter for File supports custom defined transformations defined using the WebLogic Integration Format Builder. Once defined and tested, the adapter can utilize these Message Format Language files to apply transformation to an incoming document. Chapter 4, "Service and Event Configuration," describes how to configure events and services to use Message Format Language transformation. This section provides a brief overview of how a Message Format Language template is built and tested using the Format Builder, and how this template can then be tested (once deployed in a application view) using the BEA WebLogic Adapter for File.

For more information about Message Format Language transformations, see "Building Format Definitions" in *Translating Data*:

■ For WebLogic Integration 7.0, see
  `http://edocs.bea.com/wli/docs70/diuser/fmtdef.htm`

■ For WebLogic Integration 2.1, see
  `http://edocs.bea.com/wlintegration/v2_1sp/diuser/fmtdef.htm`

To test using the sample Message Format Language file, `Sprockets_PO.mfl`, supplied with this product, follow the procedure outlined below:

1. From the Windows Start menu, choose Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Format Builder. The Format Builder window opens.

Figure 6-1   Format Builder Repository Window

2. Choose Log In from the Repository menu. The WebLogic Integration Repository window opens.

Figure 6-2   WebLogic Integration Repository Login Window

3. Enter your user name and password and click Continue.

    After successfully signing on, you can import the sample Message Format Language template.

4. Choose Open from the File menu.

Figure 6-3   Format Builder Repository
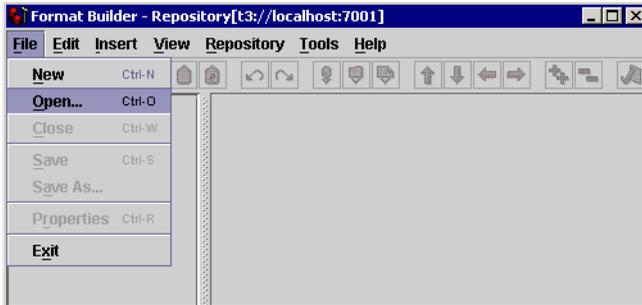
5. Select the Sprockets_PO.mfl file that is supplied in the BEA_SAMPLES.zip file.

Figure 6-4   Format Builder Repository
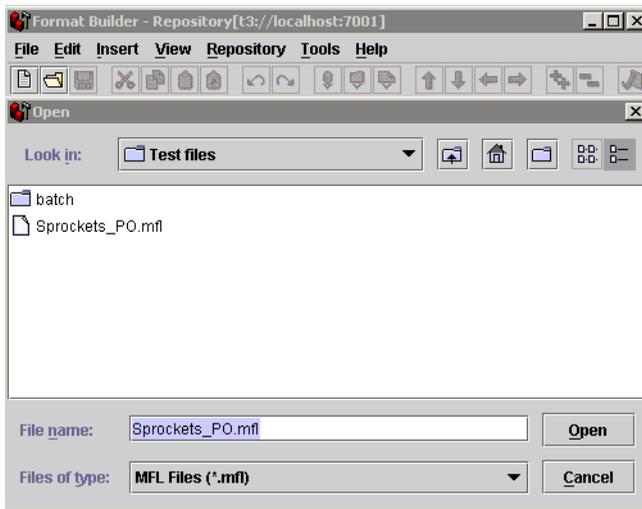
6. Double-click the PurchaseRequest envelope to expand the field definitions below and to see how field types were defined.

Figure 6-5 Format Builder
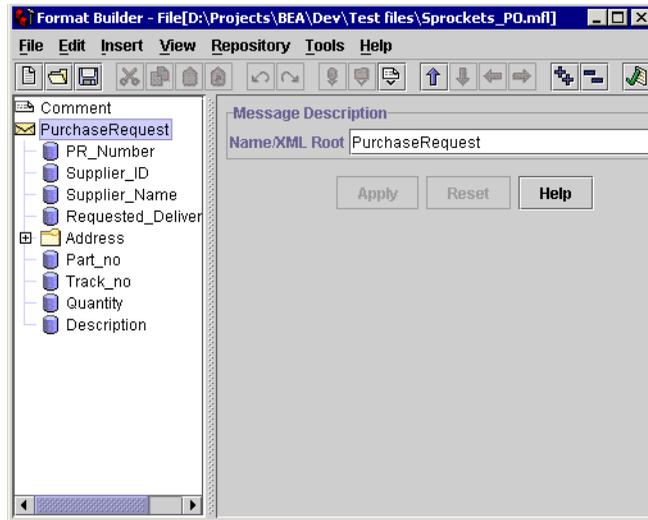
All that is required now is to store the MFL in the repository.

7. Choose Store As from the Repository menu.

Figure 6-6 Format Builder Window
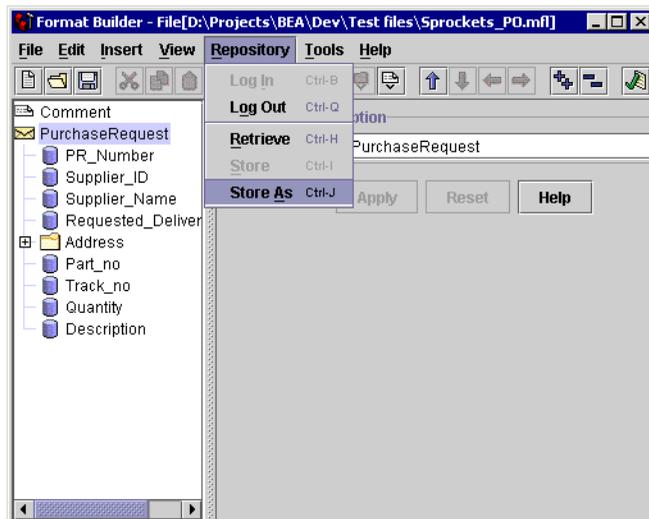
8. Give your MFL a name and store it in the root folder or any folder you desire. In this example, the MFL PurchaseOrder is stored in a folder called SprocketsRus.

Figure 6-7   Store Document Window
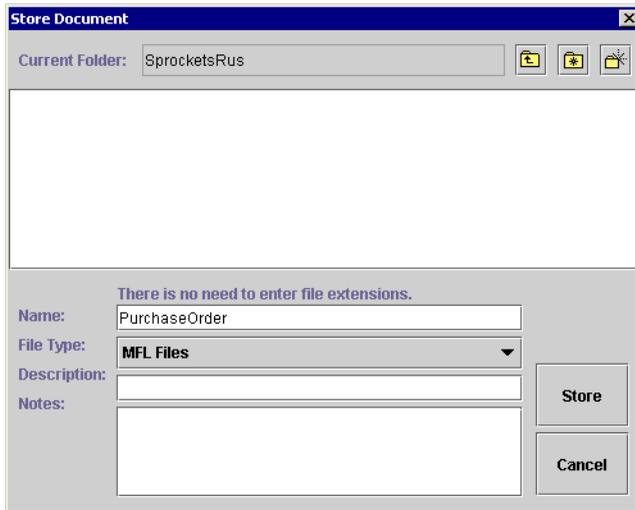
Once stored, you can test your adapter service or event by configuring as documented in Chapter 4, "Service and Event Configuration." To test this document, use the sample documents supplied in the BEA_SAMPLES.zip file. You can use Sprockets_PO.txt to test inbound transformations (for example, event and service File Read operations) and Sprockets_PO.xml to test outbound service operations. For service and event File Read tests, place the file (Sprockets_PO.txt) in the location that is being polled for the document. To test services, copy the content of Sprockets_PO.xml into the text window in the test screen.

7 Using Tracing

Tracing is an essential feature of an adapter. Most adapters integrate different applications and do not interact with end users while processing data. Unlike a front-end component, when an adapter encounters an error or a warning condition, the adapter cannot stop processing and wait for an end user to respond.

Moreover, many business applications that are connected by adapters are mission-critical. For example, an adapter might maintain an audit report of every transaction with an EIS. Consequently, adapter components must provide both accurate logging and auditing information. The adapter tracing and logging framework is designed to accommodate both logging and auditing.

This section describes tracing for services and events. It contains the following topics:

- Levels and Categories of Tracing

- Tracing and Performance

- Creating Traces for Services and Events

Levels and Categories of Tracing

Tracing is provided by both the BEA adapter framework and by each individual adapter product. The BEA WebLogic Integration framework provides five distinct levels of tracing:

Table 7-1 Tracing levels

Level	Indicates
AUDIT	An extremely important log message related to the business processing performed by an adapter. Messages with this priority are always written to the log.
ERROR	An error in the adapter. Error messages are internationalized and localized for the user.
WARN	A situation that is not an error, but that could cause problems in the adapter. Warning messages are internationalized and localized for the user.
INFO	An informational message that is internationalized and localized for the user.
DEBUG	A debug message, that is, information used to determine how the internals of a component are working. Debug messages usually are not internationalized.

The adapter framework provides three specialized categories of tracing:

Table 7-2 Tracing categories

Level	Indicates
Basic Trace	Basic traces. Displays the input XML (up to 300 bytes) before parsing, and shows the request being processed. The default setting is off.
Verbose Trace	More extensive traces. Displays configuration parameters used by the adapter. The default setting is off.

Table 7-2 Tracing categories

Level	Indicates
Document Trace	Displays the input document after it was analyzed and the response document being returned. Because some documents are very large, this trace category can severely affect performance and memory use. The default setting is off.

Note: To obtain the appropriate trace, both the level and the category must be declared. In a debug situation, BEA Customer Support will request (minimally) a Basic and a Verbose trace.

Tracing and Performance

The additional trace capabilities provided by the adapter are not strictly hierarchic; rather they are categorized. These traces are designed to provide debugging help with minimum effect on performance. All internal adapter traces are controlled through the additional tracing settings, and all additional settings route their output to the standard debug setting.

If you configure the adapter for additional settings and do not configure standard trace settings, the traces are generated but never appear in output. This affects performance, as the production of the trace continues even though you receive no benefit of the additional trace information.

Creating Traces for Services and Events

This following topics discuss the steps required to create traces to diagnose adapter problems.

Creating Traces for a Service

To create traces for a service:

1. Create or modify the service.

2. Ensure that all of the adapter parameters are entered correctly.

Figure 7-1   Add Service window

3. Select the appropriate schema from the drop-down list.

4. Select the appropriate trace levels as described in Table 7-2: Trace, Verbose trace, and Document trace.

5. Click Add to continue to the next configuration pane.

6. Click Continue to move to the next configuration pane.

   The Deploy Application View window opens.

Figure 7-2 Deploy Application View window
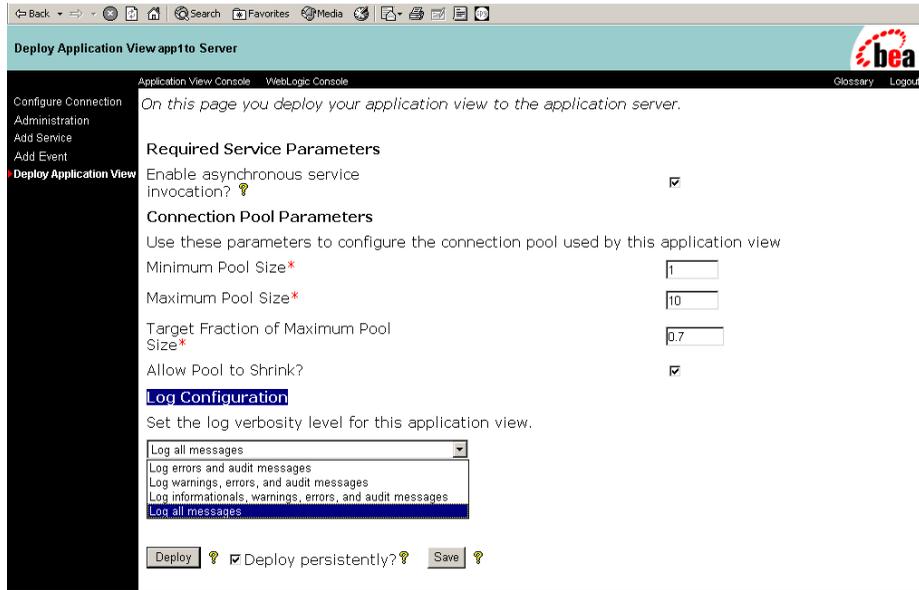
7. Navigate to the Log Configuration area and select the desired trace level.

    This pane enables you to select the trace level for the BEA WebLogic Integration framework.

    For maximum tracing, select Log all Messages. This is recommended to obtain optimum debugging information for BEA support personnel.

    Note: This causes all generated messages to be written to the log. You must select the desired category as defined in Table 7-2 in the adapter to generate the required messages.

8. Click Deploy (or Save) to set the trace settings and deploy the application view.

    Traces are created the next time the service is invoked.

    Traces are output to a file named BEA_FILE_1_0.log in the WebLogic Domain home directory.

Creating or Modifying the Tracing Level for an Event

To create or modify the WebLogic framework tracing level for an event:

1. Logon to the BEA WebLogic Server Console.

Figure 7-3   WebLogic Server Console

2. Select Web Applications.

3. Select the EventRouter corresponding to the adapter that will be traced. For example, if you require traces for a FILE event, select BEA_FILE_1_0_EventRouter.war.

4. Click Edit Application Deployment Descriptors.

5. When the following window opens, select Servlets.

Figure 7-4   WebLogic Server Console: Configuration
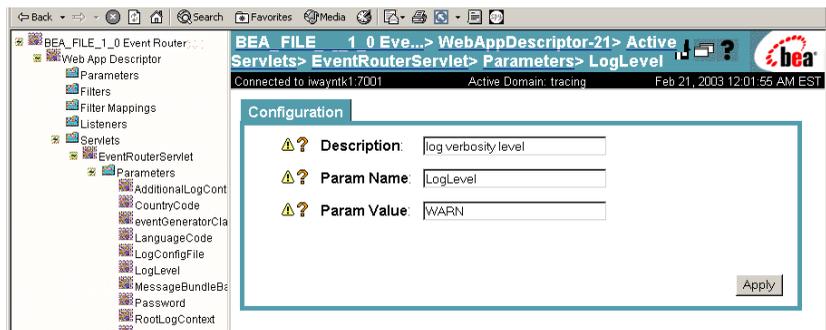
6. In the folder below Servlets, select EventRouterServlet.

7. Select Parameters.

8. Select LogLevel.

   This pane enables you to select the trace level for the BEA WebLogic Integration framework.

   For maximum tracing, enter DEBUG. This is recommended to obtain optimum debugging information for BEA support personnel.

   The following levels are valid:

Table 7-3 Tracing levels

| Level	Indicates
AUDIT	An extremely important log message related to the business processing performed by an adapter. Messages with this priority are always written to the log.
ERROR	An error in the adapter. Error messages are internationalized and localized for the user.
WARN	A situation that is not an error, but that could cause problems in the adapter. Warning messages are internationalized and localized for the user.
INFO	An informational message that is internationalized and localized for the user.
DEBUG	A debug message, that is, information used to determine how the internals of a component are working. Debug messages usually are not internationalized.

9. Click Apply to save the newly entered trace level.

10. Click the EventRouter Servlet.

11. Select Persist to apply the logging changes.

   This change need only be made once.

   It is set for all events associated with a given adapter.

12. Return to the WebLogic Server Console.

Figure 7-5   WebLogic Server Console: Redeploy
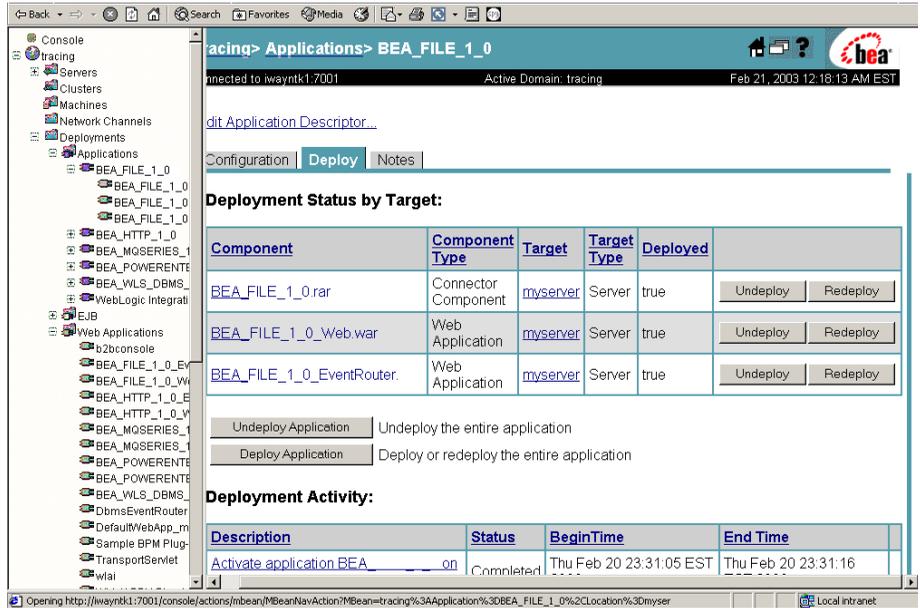
13. Select Applications from the WebLogic Server Console.

14. Select the adapter whose EventRouter you have modified in the previous steps.

15. The right pane displays the following adapter components:

- BEA_FILE_1_0.rar

- BEA_FILE_1_0.web.rar

- BEA_FILE_1_0_EventRouter.war.

16. Redeploy the EventRouter by clicking the Redeploy button to the right of BEA_FILE_1_0_EventRouter.war.

Creating Adapter Logs for an Event

To create adapter logs for an event:

1. Create or modify the event.

2. Ensure that all of the adapter parameters are entered correctly.

Figure 7-6   Add Event window
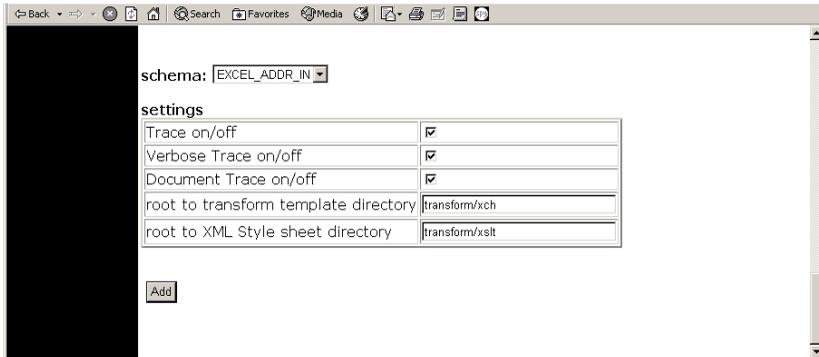
3. Select the appropriate schema from the drop-down list.

4. Select the appropriate trace levels as described in Table 7-2: Trace, Verbose trace, and Document trace.

5. Click Add to continue to the next configuration pane.

6. Click Continue to move to the next configuration pane.

The Deploy Application View window opens.

Figure 7-7   Deploy Application View window
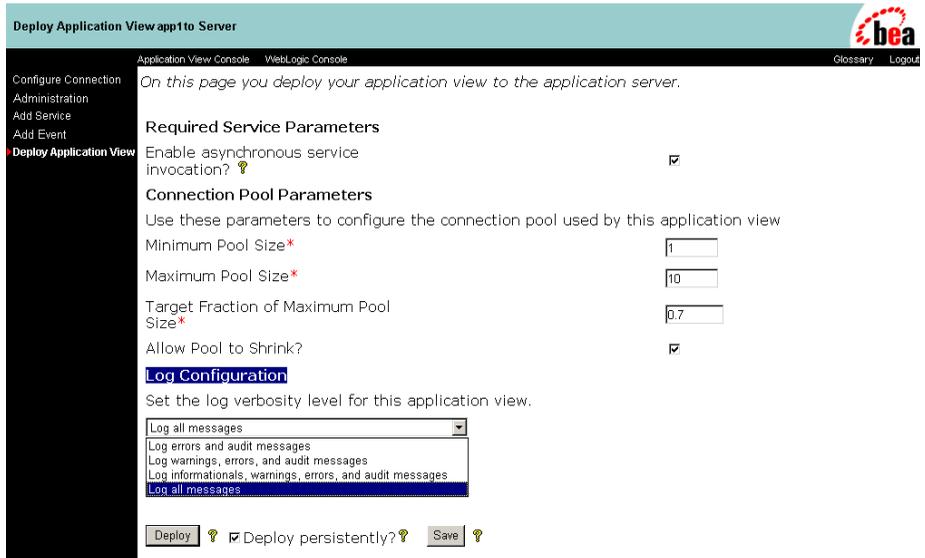
7. Navigate to the Log Configuration area and select the desired trace level.

   This pane enables you to select the trace level for the BEA WebLogic Integration framework.

   For maximum tracing, select Log all Messages. This is recommended to obtain optimum debugging information for BEA support personnel.

8. Click Deploy (or Save) to set the trace settings and deploy the application view.

   Traces are created the next time the event occurs.

   Traces are output to a file named BEA_FILE_1_0.log in the WebLogic Domain home directory.