BEA WebLogic Adapter for Email

User Guide

Release 7.0 with Service Pack 2
Document Date: March 2003

Copyright

Restricted Rights Legend

Trademarks or Service Marks

BEA WebLogic Adapter for Email User Guide

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

Table of Contents

4. Service and Event Configuration

5. BEA WebLogic Adapter for Email Integration Using Studio

6. Transforming Document Formats

7. Using Tracing

About This Document

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

- Chapter 1, "Introducing the BEA WebLogic Adapter for Email," introduces the BEA WebLogic Adapter for Email, 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 Email," describes 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 Email Integration Using Studio," describes how events are incorporated into workflow design.
- Chapter 6, "Transforming Document Formats," describes how to use Message Format Language (MFL) 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 Email and other applications. It describes how to use the BEA WebLogic Adapter for Email 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 Email Installation and Configuration Guide*
- *BEA WebLogic Adapter for Email 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 Email 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 Email documentation.

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

If you have any questions about this version of the BEA WebLogic Adapter for Email, or if you have problems using the BEA WebLogic Adapter for Email, 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.

CHAPTER

1 Introducing the BEA WebLogic Adapter for Email

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

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.

E-mail is a vital tool in an enterprise for the exchange of information. The e-mail protocol must be taken into consideration when planning an integration strategy. The BEA WebLogic Adapter for Email incorporates the SMTP protocol with the POP3 protocol to optimize the integration files with enterprise application systems. The BEA

WebLogic Adapter for Email enables integration with e-mail messages using the Simple Mail Transport Protocol (SMTP) communications protocol. The e-mail messages can be XML, non-XML ASCII, or custom data formats. This provides a convenient and simple method for integrating with application systems using BEA WebLogic Integration.

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

- Asynchronous, bi-directional message interactions between BEA WebLogic Integration and SMTP/POP3 servers.
- A business process that runs within BEA WebLogic Integration to transfer data to and from SMTP 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 (for Excel, the adapter converts for inbound only). 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. These formats are supported either in the body of the e-mail or in an attachment file.
- Events that can be routed through the SMTP messaging system. E-mail messages may contain proprietary custom data formats that need to be transformed. The BEA WebLogic Adapter for Email supports processing of XML, ASCII (CSV, CDF, and Excel) and custom non-XML based messages containing structured, binary, and string data.

How the BEA WebLogic Adapter for Email Works

The adapter provides transport protocol support so that it can “listen” and “emit” documents using the POP3/SMTP protocol.

The listening capability has been implemented as an event within WebLogic Integration. The event can be configured to act as an e-mail client by supplying an e-mail address, password, e-mail host and so on, along with a number of 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. The BEA WebLogic Adapter for Email ensures that any incoming document can be converted to the XML format dictated by your event or service schemas. The BEA WebLogic Adapter for Email 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 a number of options that are configured with the BEA Application Explorer and WebLogic Application View Console:

- **Transformation services.** Outbound documents can be transformed to convert XML documents into non-XML formats. The BEA WebLogic Adapter for Email can also be used in conjunction with other BEA WebLogic Adapters to process a variety of message types, such as SAP IDocs, 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 e-mail 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

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 Email 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 Email.

The BEA WebLogic Adapter for Email 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 workflows, 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 Email 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 supported operations.

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>

   <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 Email."

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, EMAIL 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 EMAIL, and the connection name is EMAILDev, then the schema repository is the directory:

```
/usr/opt/bea/bse/EMAIL/EMAILDev
```

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 related 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 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 Email) 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 Email.
- `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 Email and WebLogic Integration, you should set up four subdirectories to store the schemas; the subdirectories should be in a parent `Email` directory:

```
D:\TraderSystems\BEAapps\Email\FTPprod
D:\TraderSystems\BEAapps\Email\FTPdev
D:\TraderSystems\BEAapps\Email\FTPuat

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 2-7.

Listing 2-3 Instance Document for OrderIn Event

```
<?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 Email 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 Email 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 Email 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.

CHAPTER

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

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

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 Email, 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. Information in these disparate systems can be effectively shared in the IT environment for an organization.

Information stored in Excel documents can be easily used to update data stored in SAP. With the BEA WebLogic Adapter for Email, an Excel document embedded in an e-mail can trigger an event after being automatically converted to XML. Documents can be emitted in XML or non-XML format using the adapter.

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 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 needs 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.mf` 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 must be stored in the `transform/dic` directory. For more information on file locations, see the *BEA WebLogic Adapter for Email 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 a new application view:

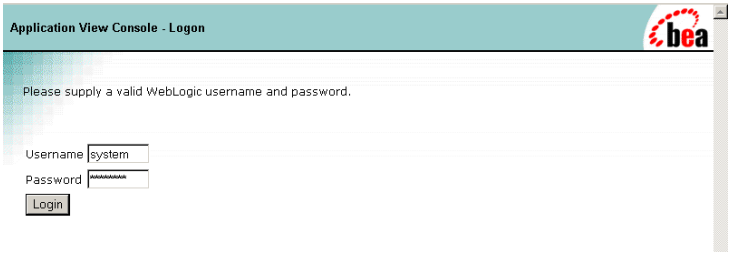
1. Open the Application View Console, which is found at the following location:

   `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

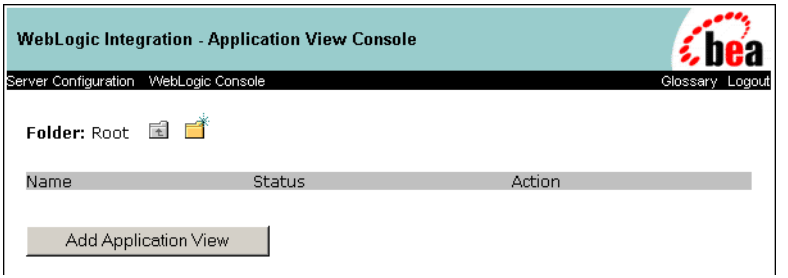


Note: If the user name is not `system`, it must be included in the `adapter` group. For more information on adding the administrative server user name to the `adapter` group, see the *BEA WebLogic Adapter for Email Installation and Configuration Guide*.

3. Click Login.

   The WebLogic Integration Application View Console opens.

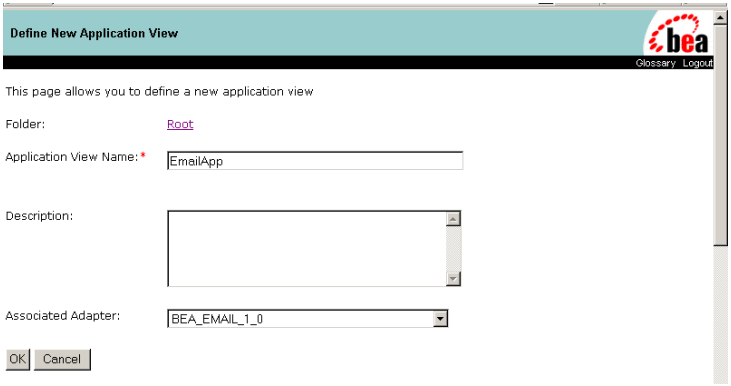
Figure 3-2 Application View Console Window



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



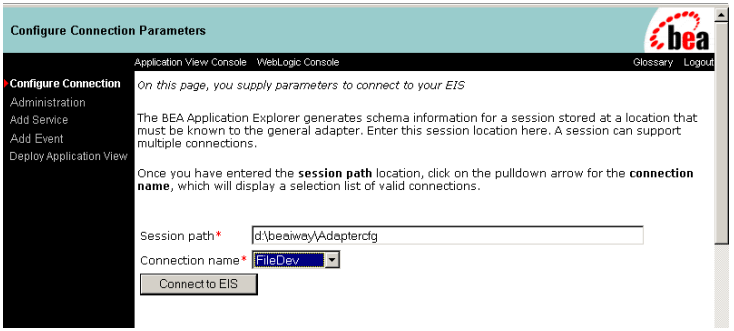
5. In the Application View Name field, enter a name. The name should describe the set of functions performed by this application. Each application view name must be unique to its adapter. Valid characters are a-z, A-Z, 0-9, and _ (underscore).

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_EMAIL_1_0` to associate the BEA WebLogic Adapter for Email with this application view.

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

Figure 3-4 Configure Connection Parameters Window



The BEA Application Explorer is used to explore and present metadata relating to the BEA WebLogic Adapters. The BEA Application Explorer creates and exports the schemas required by WebLogic Integration Server to configure services and events. This metadata is stored in a file system (definable by the application designer) and needs to be supplied to the application view setup window. See "Schemas, Dictionaries, and Transformation Templates" on page 3-2.

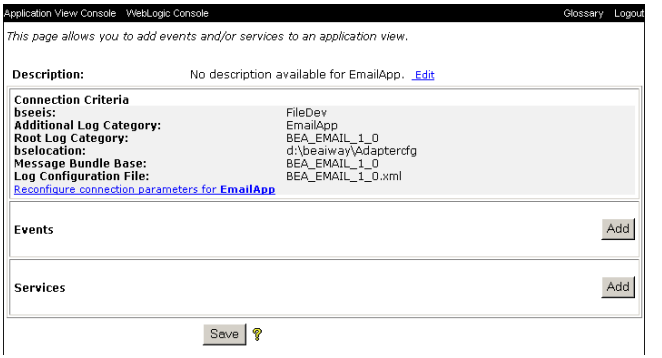
9. Enter the root directory containing your schema subdirectories. For example, `d:\bea\emailadapter`.

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

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

11. 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."

CHAPTER

4 Service and Event Configuration

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

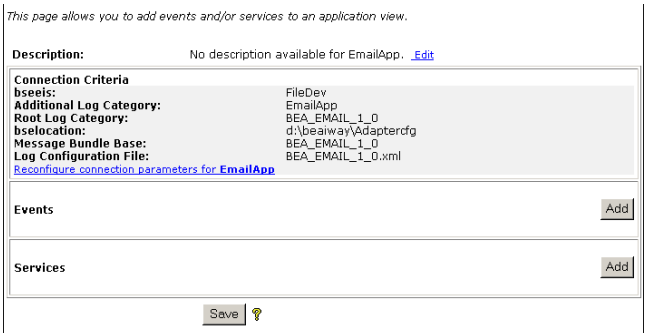
Adding a Service to an Application View

After you create and configure an application view as described in Chapter 3, "Defining an Application View for the BEA WebLogic Adapter for Email," 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.

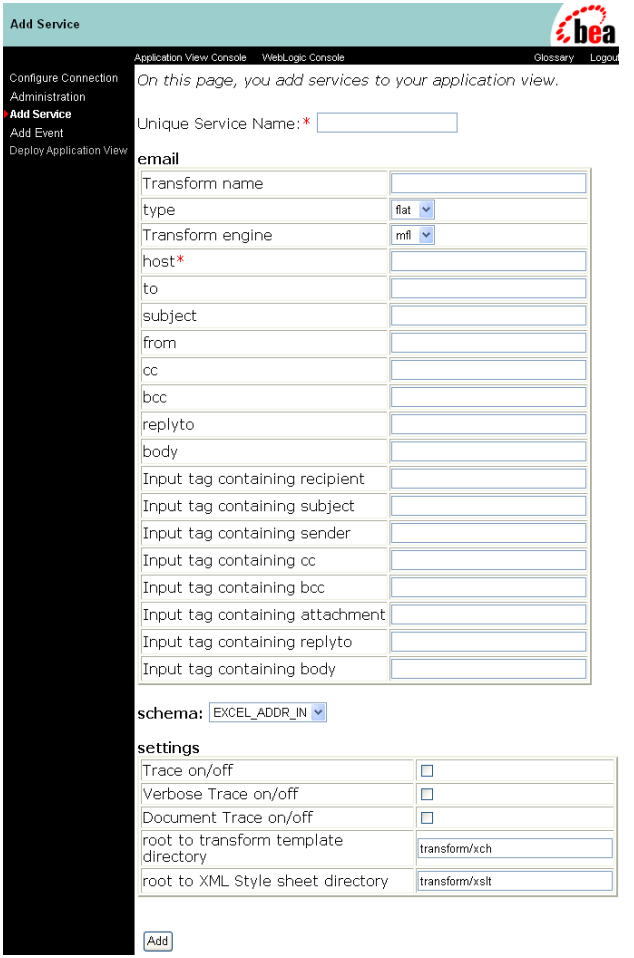
Figure 4-1 Application View Console Administration Window



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. lEnter the required settings (required fields are marked with an asterisk) to configure your process.

Descriptions of the email service parameter settings follow:

Table 4-1 Email service parameters

Setting	Properties/Description
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. **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:** String **Description:** Output type of the e-mail (flat for non-XML, XML for XML).
Transform engine	**Type/Value:** Drop-down list **Description:** Engine required to transform the document (Message Format Language (mfl), XSL Transformation (xslt), Supplied transformation templates (xch)).
host* (*Required)	**Type/Value:** String **Description:** SMTP host DNS name.
to	**Type/Value:** String **Description:** E-mail address of the intended recipient.
subject	**Type/Value:** String **Description:** Title of e-mail to be sent.
from	**Type/Value:** String **Description:** E-mail address of the sender.
cc	**Type/Value:** String **Description:** E-mail address of the carbon copy recipient.
bcc	**Type/Value:** String **Description:** E-mail address of the blind carbon copy recipient.
replyto	**Type/Value:** String **Description:** E-mail address to which replies will be sent.

Table 4-1 Email service parameters

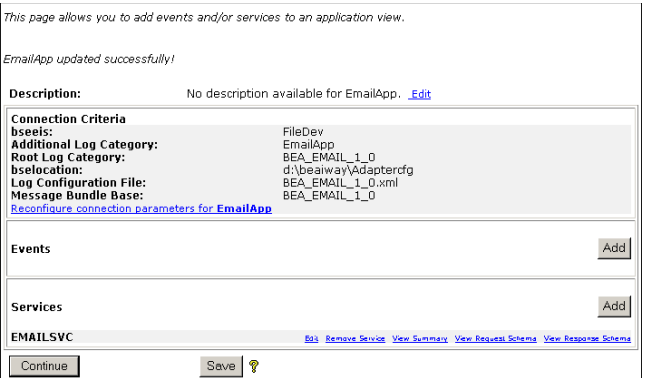
Setting	Properties/Description
body	**Type/Value:** String **Description:** Body of the e-mail message.
Input tag containing recipient	**Type/Value:** String **Description:** XML tag which identifies the e-mail address of the intended recipient. This setting is valid only if the type setting is `XML`. The data in this XML tag overrides the `to` setting.
Input tag containing subject	**Type/Value:** String **Description:** XML tag which identifies the title of the e-mail address to be sent. This setting is valid only if the type setting is `XML`. The data in this XML tag overrides the `subject` setting.
Input tag containing sender	**Type/Value:** String **Description:** XML tag which identifies the e-mail address of the sender. This setting is valid only if the type setting is `XML`. The data in this XML tag overrides the `from` setting.
Input tag containing cc	**Type/Value:** String **Description:** XML tag which identifies the e-mail address of the carbon copy recipient. This setting is valid only if the type setting is `XML`. The data in this XML tag overrides the `cc` setting.
Input tag containing bcc	**Type/Value:** String **Description:** XML tag which identifies the e-mail address of the blind carbon copy recipient. This setting is valid only if the type setting is `XML`. The data in this XML tag overrides the `bcc` setting.
Input tag containing attachment	**Type/Value:** String **Description:** XML tag which identifies the element of the XML document that contains the data that will constitute the attachment.
Input tag containing replyto	**Type/Value:** String **Description:** XML tag which identifies the e-mail address required for responses. The data in this XML tag overrides the replyto setting.
Input tag containing body	**Type/Value:** String **Description:** XML tag that will contain the body of the email text.

Table 4-1 Email service parameters

Setting	Properties/Description
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.

7. Select the required schema at the bottom of the window, from the schema drop-down list.
8. Click Add. The Application View Administration window opens.

Figure 4-3 Application View Administration Window



9. Click Continue or Save.

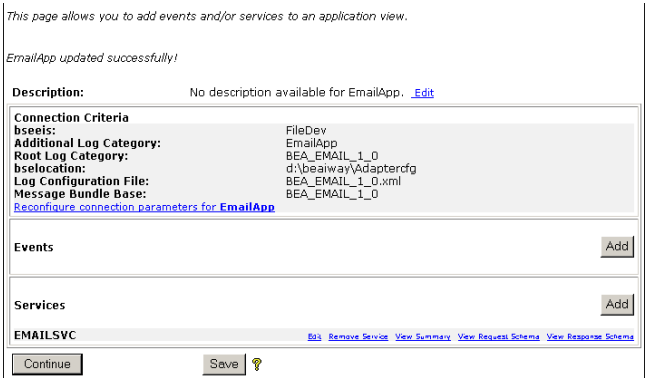
   At this point, the application can be deployed or more services or events can be configured. Once the application has been deployed, you can test the service. To deploy the application, see "Deploying an Application View" on page 4-13. To test the application, see "Testing a Service or Event" on page 4-15.

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 Email 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 Email."

1. If your application is deployed, undeploy it and edit the application view.

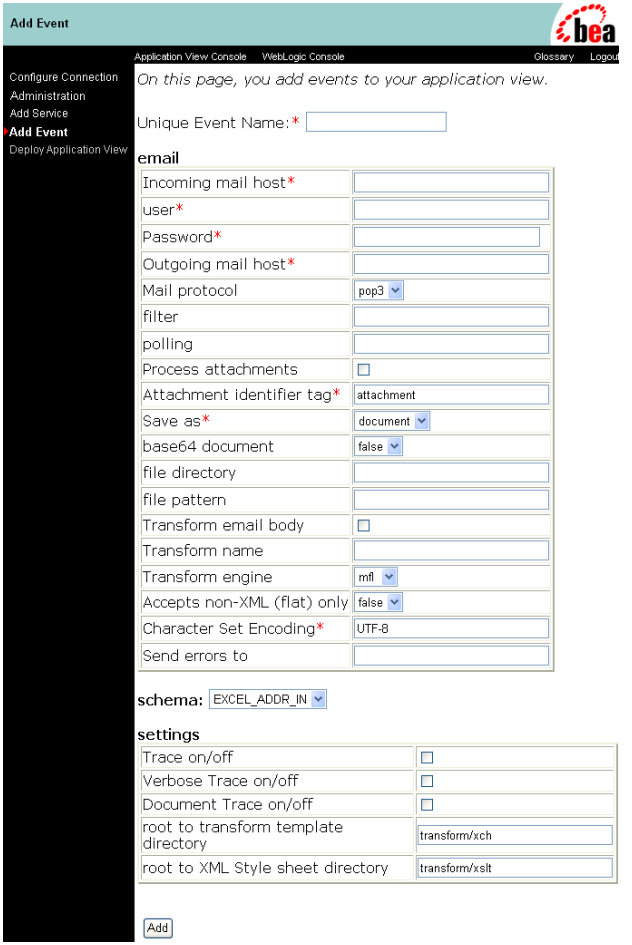
Figure 4-4 Application View Console Administration Window



2. From the Application View Console Administration window, click Add in the Events section of the administrative pane.

The Add Event window opens.

Figure 4-5 Add Event Window



3. Enter a Unique Event Name.
4. Enter the email settings (required fields are marked with an asterisk) to configure your process.

Descriptions of the email configuration settings follow.

Figure 4-6 Email Event Properties

Setting	Properties/Description
Incoming mail host* (*Required)	**Type/Value:** String **Description:** Incoming mail (POP3/IMAP) server.
user* (*Required)	**Type/Value:** String **Description:** User name at the host to receive e-mail. This is also the ID that will be used to log into the e-mail server.
Password* (*Required)	**Type/Value:** String **Description:** Password for e-mail account.
Outgoing mail host* (*Required)	**Type/Value:** String **Description:** Outgoing mail SMTP server.
Mail_protocol	**Type/Value:** String **Description:** Incoming mail server protocol.
filter	**Type/Value:** String **Description:** Picks up e-mails with the subject specified.
polling	**Type/Value:** Duration, using the format *xx*H:*xx*M:*xx*S. For example: 1H:2M:3S is 1 hour, 2 minutes, and 3 seconds. **Description:** The maximum wait interval between checks for new email. 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 respond to a stop command. If timeout is set to 0, the listener runs once and terminates. Default is 2 seconds.
Process attachments	**Type/Value:** Check box **Description:** Configures event to process attachments.
Attachment identifier tag* (*Required)	**Type/Value:** String **Description:** XML tag which identifies the attachment. **Note:** Since the attachment information can change the incoming message, the schema must take this into account.

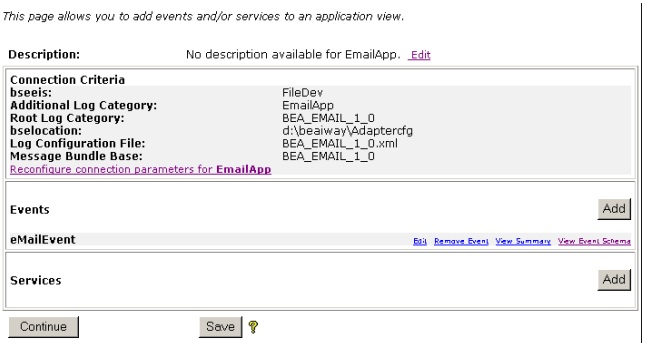
Setting	Properties/Description
Save as* (*Required)	**Type/Value:** Drop-down list **Description:** Determines what format the attachment will be in after processing. **Values:** ■ Document - This determines that the attachment will be read and embedded in the incoming XML document. ■ File - This configures the event to store the attachment in a file system.
base64 document	**Type/Value:** String **Description:** Specifies whether the attachments are to be base64 encoded. **Values:** true or false
file directory* (*Required if Save_as is file)	**Type/Value:** String **Description:** Directory to which files are written.
file pattern* (*Required if Save_as is file)	**Type/Value:** String **Description:** File name pattern.
Transform email body	**Type/Value:** Check box **Description:** Check this box if transformation of the body of the text in the email is required.
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.
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).
Accept non-XML (flat) only	**Type/Value:** Check box **Description:** Check this box if the body of the text in the email is non-XML and is being transformed.

Setting	Properties/Description
Character Set Encoding* (*Required)	**Type/Value:** String **Description:** The character encoding system to use. It defaults to UTF-8.
Send errors to	**Type/Value:** String **Description:** Directory path indicating where the error messages will be placed in the case of a failure.
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.

5. Select the required schema from the schema drop-down list and click Add.
6. Click Add.

   After adding the event process, the following window opens.

Figure 4-7 Application View Administration Window



7. Click Save to save your settings. You can deploy your application view (complete with configured events and/or services) by following the steps described in "Deploying an Application View" on page 4-13. Then you can test your application view by following the steps described in "Testing a Service or Event" on page 4-15.

Deploying an Application View

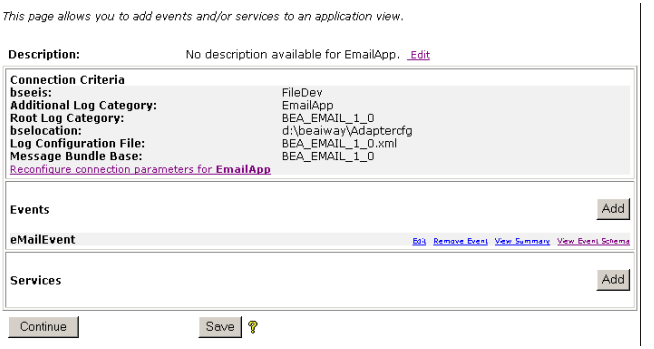
You may deploy an application view when you have added at least one event or service to it. You must deploy an application view before you can test its services and events or use the application view in the WebLogic Server environment. Application view deployment places relevant metadata about its services 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`

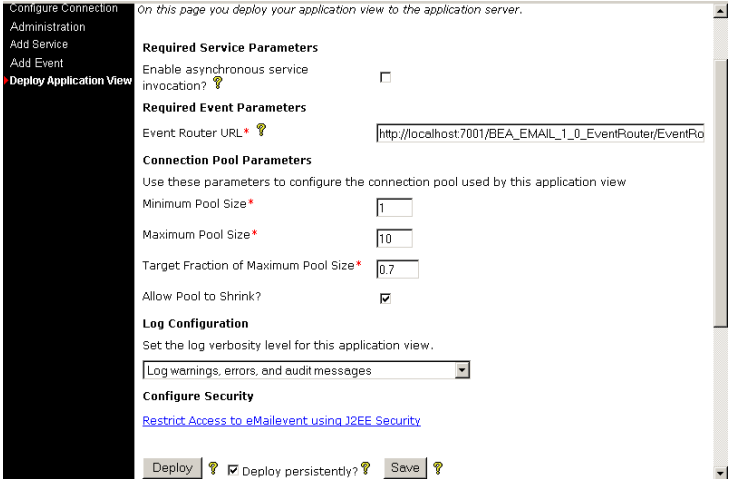
Figure 4-8 Application View Console Administration Window



2. From the Application View Console Administration window, click Continue.

   The Deploy Application View window opens.

Figure 4-9 Deploy Application View Window

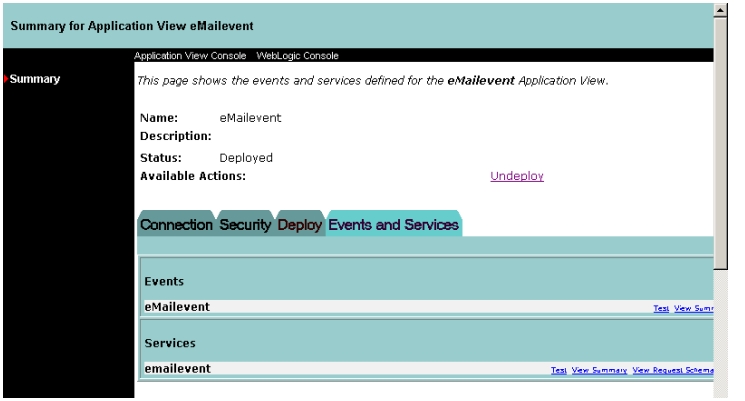


Note: To enable business process management functionality or other authorized clients to asynchronously call the services (if any) of this application view, select Enable asynchronous service invocation.

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

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

Figure 4-10 Summary for Application View Window



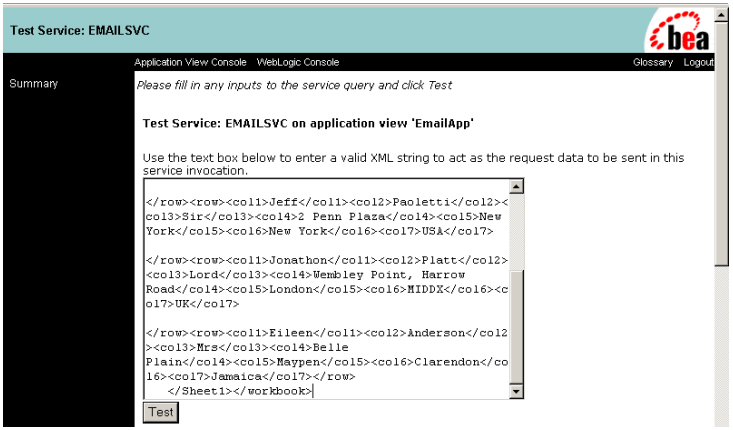
After you create and deploy an application view, you can test the services and events. For more information, see "Testing a Service or Event" on page 4-15.

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.

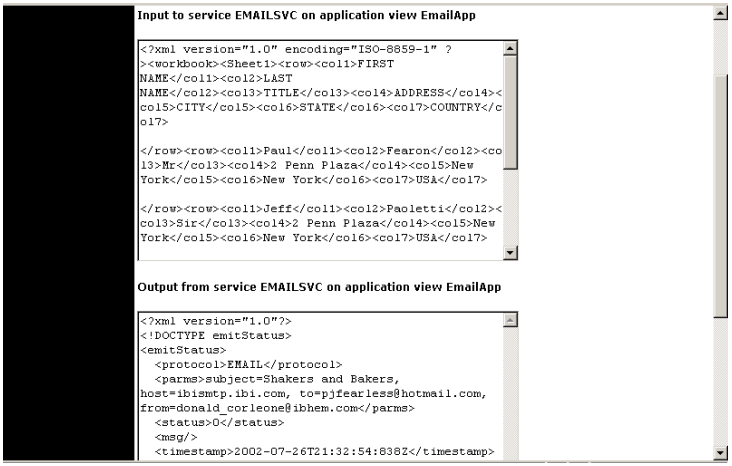
Figure 4-11 Test Service Window



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 or event 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`

Email Attachments

The way an attachment is handled relies on factors such as the format of the body of the email. For instance, if the body of the email and the attachment to the email are in XML form, then the attached document can be merged with the body of the email and the new XML document can then be passed onto the workflow.

If the body of the email is in XML form, but the attached document is not or if the body of the email is not in XML form, then the attached document can be written to a file system, to be obtained and processed by another relevant adapter (for example the BEA WebLogic Adapter for File). The body of the email can still be transformed (if required).

Configuration options are available for all possibilities in the Service or Event configuration JSP pages.

CHAPTER

5 BEA WebLogic Adapter for Email Integration Using Studio

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

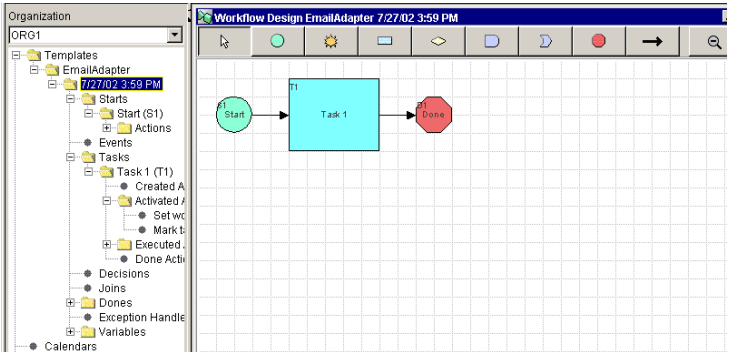
- Business Process Management Functionality

Business Process Management Functionality

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

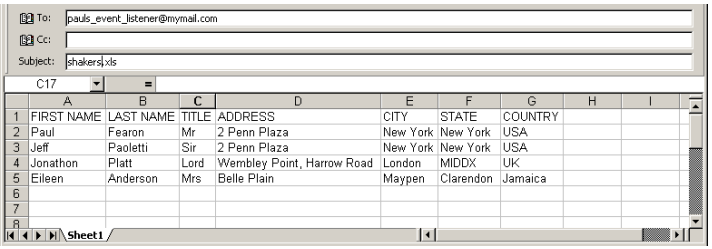
The following window depicts a simple workflow design to be triggered by an event described in Chapter 3, "Defining an Application View for the BEA WebLogic Adapter for Email." The incoming event converts a document in e-mail form and utilizes a service to propagate the event to a workflow (for example, an Excel file taken from an e-mail is converted to XML and placed in a local file system). The event responds to the e-mail with an embedded Excel document, then converts the Excel document into XML format and propagates it (using the workflow) to the service.

Figure 5-1 Workflow Design Window



The following window depicts the sample Excel document being used in this process. This Excel file is contained in a zip file called `BEA_EMAIL_INSTALL.ZIP`, which is supplied with your installation package.

Figure 5-2 Original Excel Document Window



	A	B	C	D	E	F	G	H	I
1	FIRST NAME	LAST NAME	TITLE	ADDRESS	CITY	STATE	COUNTRY		
2	Paul	Fearon	Mr	2 Penn Plaza	New York	New York	USA		
3	Jeff	Paoletti	Sir	2 Penn Plaza	New York	New York	USA		
4	Jonathon	Platt	Lord	Wembley Point, Harrow Road	London	MIDDX	UK		
5	Eileen	Anderson	Mrs	Belle Plain	Maypen	Clarendon	Jamaica		
6									
7									
8									

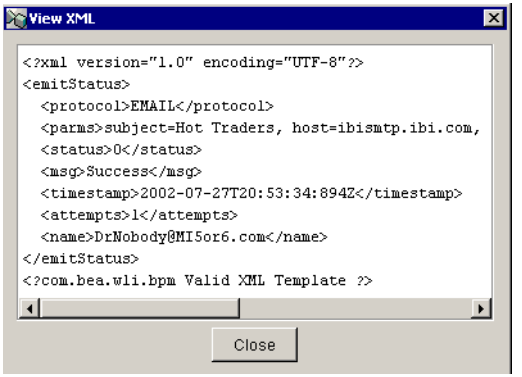
Starting with the original document, this process consists of the following steps:

1. The document is placed in an e-mail and sent to a recipient mailbox (for which the BEA WebLogic Adapter for Email event is configured), to be triggered by the arrival of the Excel document. The event has been pre-configured to convert the document from Excel to XML (using the preparse parameter).

   Once the adapter completes the process, business process management workflow then propagates the XML document onto the BEA WebLogic Adapter for Email service (configured to send the XML document to an e-mail address). 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, or HIPAA).

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

Figure 5-3 Emission Report Window



3. The resulting XML file can then be used with back-end systems that may be expecting an XML document embedded in an e-mail configured by the BEA WebLogic Adapter for Email service.

Figure 5-4 XML File Window

6 Transforming Document Formats

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

- Message Format Language Transformations

Message Format Language Transformations

The BEA WebLogic Adapter for Email 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 3, "Defining an Application View for the BEA WebLogic Adapter for Email," describes how to configure events and services to use a 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 Email.

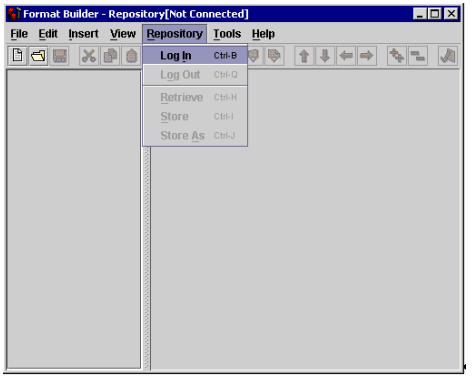
For more information about Message Format Language transformations, see the "Building Format Definitions" in *Translating Data*:

- 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 supplied with this product (`Sprockets_PO.mfl`), follow the procedure outlined below:

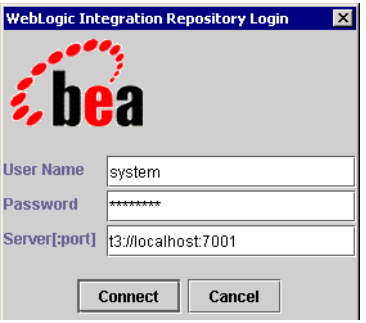
1. From the Windows Start menu, choose Programs→BEA WebLogic E-Business Platform→WebLogic Integration 2.1→Format Builder. The Format Builder window opens.

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

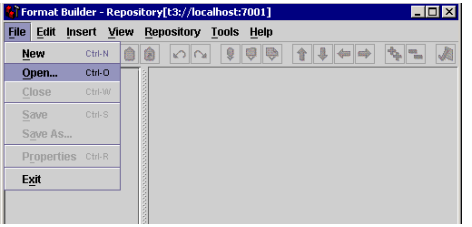


3. Enter your user name and password and click Continue.

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

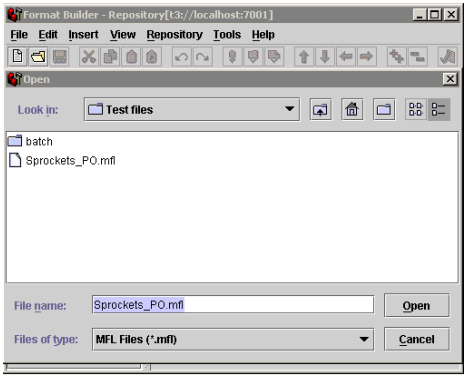
4. Choose Open from the File menu.

Figure 6-3 Format Builder - Repository Window



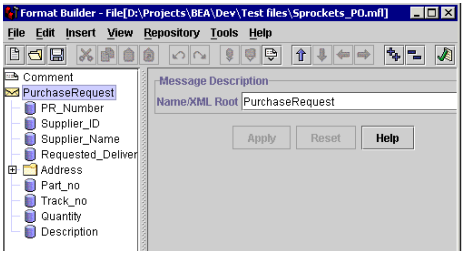
5. Select the `Sprockets_PO.mfl` file that is supplied in the `BEA_SAMPLES.zip` file.

Figure 6-4 Format Builder - Repository Window



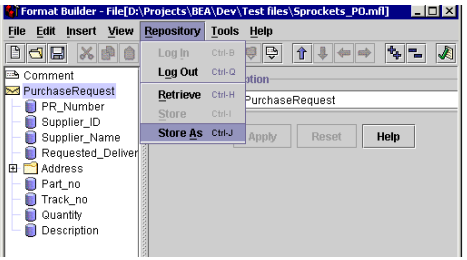
6. Double-click the PurchaseRequest envelope to expand the field definitions below and to see how field types are defined.

Figure 6-5 Format Builder Window



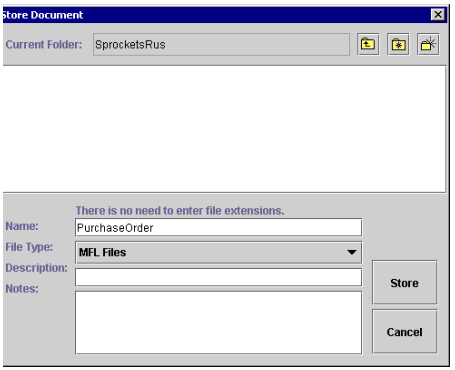
7. Choose Store As from the Repository menu.

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 you have stored the adapter service or event, you can test it by configuring it as documented in Chapter 3, "Defining an Application View for the BEA WebLogic Adapter for Email." To test this document, use the sample documents supplied in the `BEA_SAMPLES.zip` file. `Sprockets_PO.txt` can be used to test inbound transformations (for example, event and service File Read operations) or `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. For service tests, the contents of the `Sprockets_PO.xml` must be copied and pasted into the text window in the test screen.

CHAPTER

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 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 Trace 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 Trace 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 Trace 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

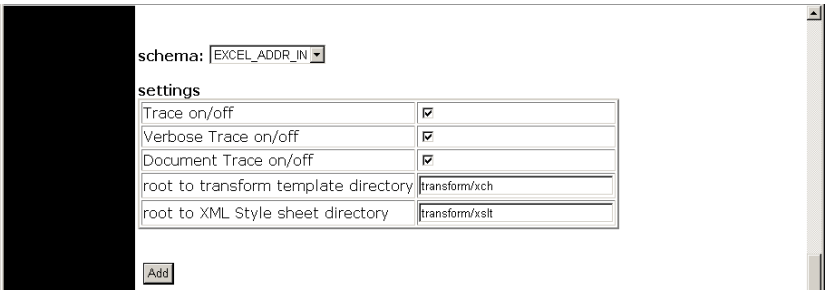
This following topics discuss the steps required to create traces to diagnose adapter problems.

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.

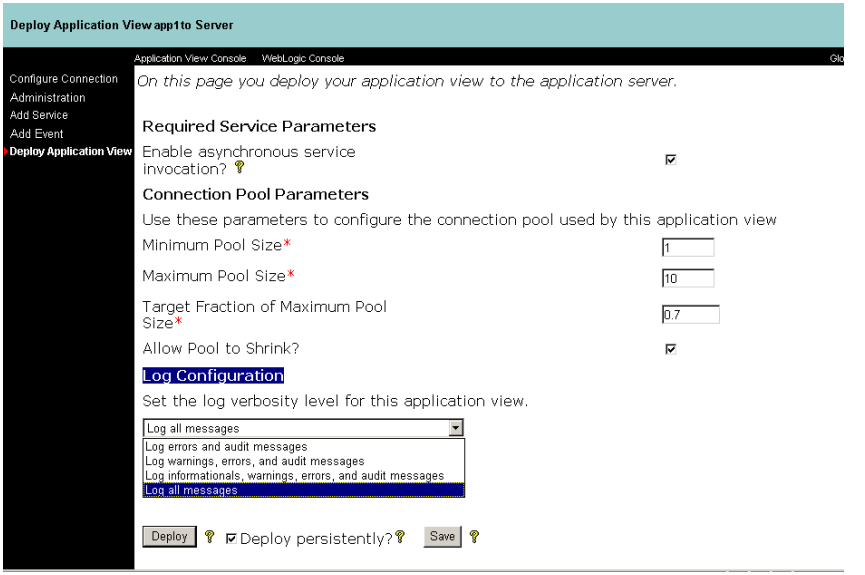
Figure 7-1 Add Service window



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.

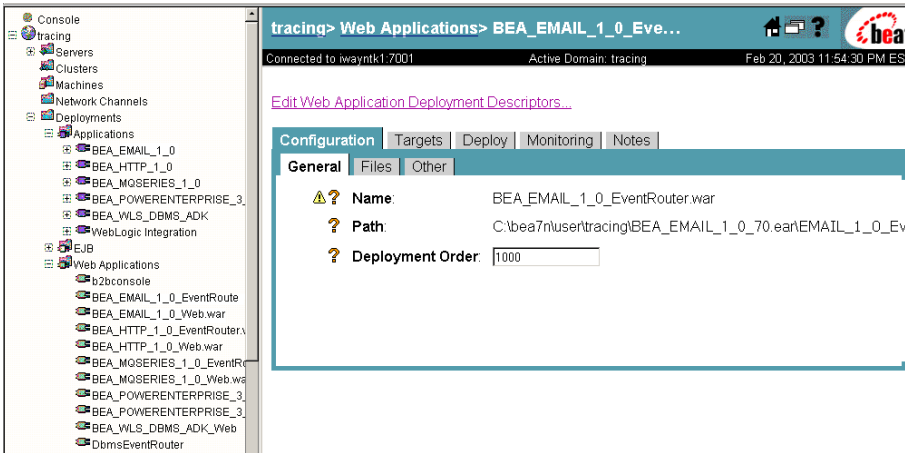
   Traces are output to a file named BEA_EMAIL_1_0.log in the WebLogic Domain home directory.

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.

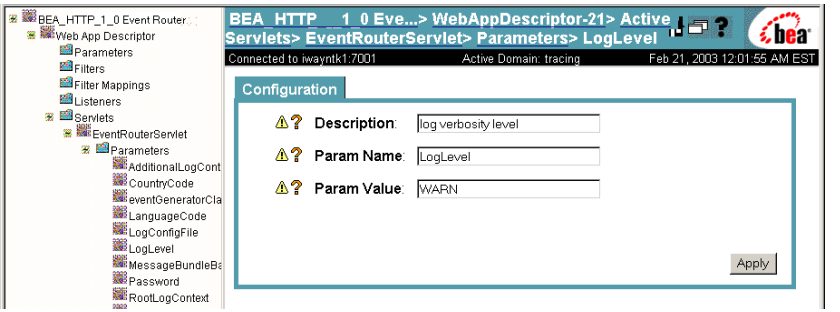
Figure 7-3 WebLogic Server Console



2. Select Web Applications.
3. Select the EventRouter corresponding to the adapter that will be traced. For example, if you require traces for an EMAIL event, select BEA_EMAIL_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 Trace 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.

Table 7-3 Trace levels

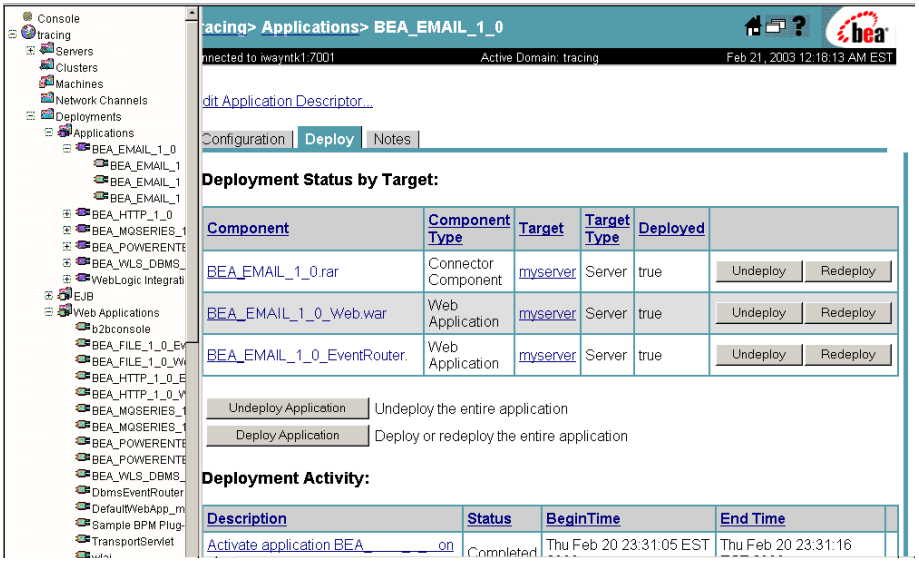
Level	Indicates
DEBUG	A debug message, that is, information used to determine how the internals of a component are working. Debug messages usually are not internationalized.

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.

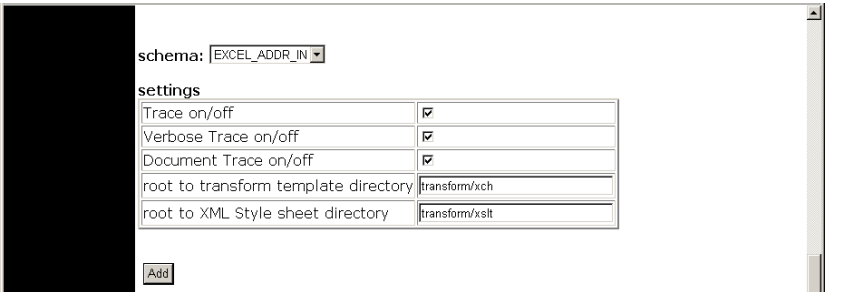
15. The right pane displays the following adapter components:
    - BEA_EMAIL_1_0.rar
    - BEA_EMAIL_1_0.web.rar
    - BEA_EMAIL_1_0_EventRouter.war.
16. Redeploy the EventRouter by clicking the Redeploy button to the right of BEA_EMAIL_1_0_EventRouter.war.

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.

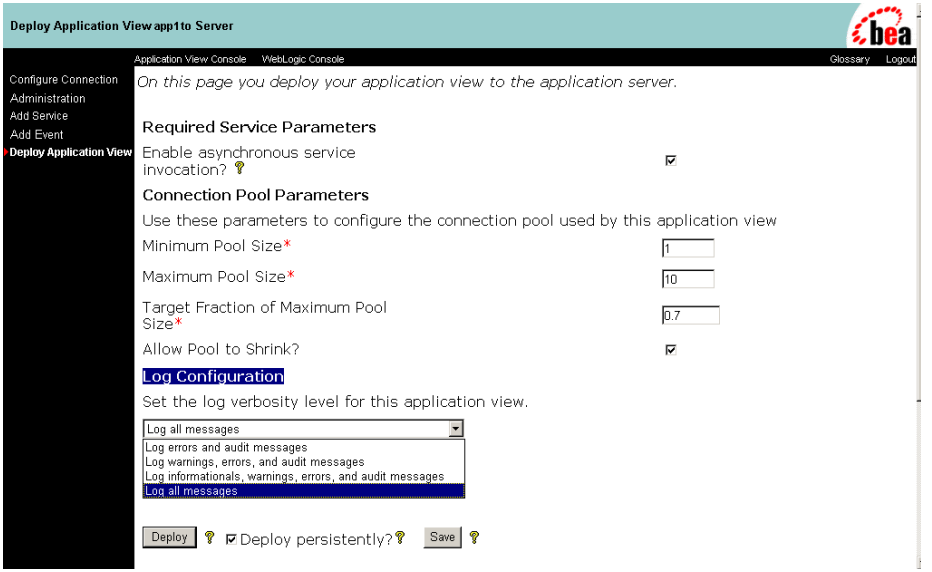
Figure 7-6 Add Event window



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_EMAIL_1_0.log in the WebLogic Domain home directory.