



BEA WebLogic Adapter for .NET[®]

User Guide

Version 8.1.2
Document Revised: January 2004

Copyright

Copyright © 2004 BEA Systems, Inc. All Rights Reserved.

Portions Copyright © 2004 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 Liquid Data for WebLogic, 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.

Contents

About This Document

Who Should Read This Documentation	vii
Additional Information	viii
How to Use This Document	ix
Contact Us!	ix
Documentation Conventions	xi

1. Introducing the BEA WebLogic Adapter for .NET

About the BEA WebLogic Adapter for .NET	1-1
About the .NET Framework	1-2
.NET Assemblies	1-2
Custom Attributes	1-3
Microsoft Message Queue	1-3
About the Architecture of the BEA WebLogic Adapter for .NET	1-4
Supported .NET Operations for Application Integration	1-5
Supported Services	1-5
Supported Events	1-5
Benefits of the Adapter for .NET	1-6
Getting Started With the Adapter for .NET	1-6
Step 1: Design the Application Integration Solution	1-7
Step 2: Determine the Required .NET Business Workflows	1-7
Step 3: Generate Schemas for .NET Integration Objects	1-8

Step 4: Define Application Views and Configure Services and Events.	1-8
Step 5: Integrate Your Application With Other BEA Software Components	1-9
Step 6: Deploy the Solution to the Production Environment.	1-9

2. Generating Schemas for .NET Integration Objects

Before You Begin	2-2
About Creating Schemas.	2-2
About the Types of Schemas You Must Generate	2-2
Service Requests.	2-3
Service Responses.	2-3
Events	2-3
About the BEA Application Explorer	2-3
About the Process for Defining Schemas Using the BEA Application Explorer . . .	2-4
Configuring Your .NET Application for Application Explorer Inquiry	2-4
Starting the BEA Application Explorer	2-7
Setting the Session Path	2-8
Managing .NET Connections	2-8
Creating a New Connection.	2-9
Using an Existing Connection	2-9
Disconnecting from .NET	2-10
Removing Connections	2-10
Creating Schemas for Services Using the BEA Application Explorer	2-10
Creating Schemas for Services and Events Manually.	2-12
About Schema Repositories.	2-13
Creating a Schema Repository.	2-13
Naming Schema Repositories.	2-13
Creating a Manifest.xml File	2-14
Creating Schemas.	2-15

Configuring Schemas for MSMQ	2-16
Sample Schema File	2-16
Removing Schemas in the BEA Application Explorer	2-20
Next Steps	2-20

3. Defining Application Views for .NET

How to Use This Document	3-2
Before You Begin	3-2
About Application Views	3-3
About Defining Application Views.	3-3
Defining Service Connection Parameters	3-5
Setting Service Properties	3-6
DotNet Service	3-7
MSMQ Service.	3-8
Common Service and Event Settings	3-9
Setting Event Properties	3-10
MSMQ Event	3-10
Defining Event Connection Parameters.	3-12
Testing Services.	3-13
Testing a dotNetService	3-14
Testing an MSMQ Service	3-16
Testing Events Using a Service.	3-19

Index

About This Document

This document describes how to use the BEA WebLogic Adapter for .NET. This document is organized as follows:

- [Chapter 1, “Introducing the BEA WebLogic Adapter for .NET,”](#) describes the adapter, how it relates to both .NET objects and WebLogic Integration.
- [Chapter 2, “Generating Schemas for .NET Integration Objects,”](#) describes how to generate schemas for your .NET objects.
- [Chapter 3, “Defining Application Views for .NET,”](#) describes application views and how to use them to configure events and services.

Who Should Read This Documentation

This document is intended for the following members of an integration team:

- **Integration Specialists**—Lead the integration design effort. Integration specialists have expertise in defining the business and technical requirements of integration projects, and in designing integration solutions that implement specific features of WebLogic Integration. The skills of integration specialists include business and technical analysis, architecture design, project management, and WebLogic Integration product knowledge.
- **Technical Analysts**—Provide expertise in an organization’s information technology infrastructure, including telecommunications, operating systems, applications, data repositories, future technologies, and IT organizations. The skills of technical analysts include technical analysis, application design, and information systems knowledge.

- Enterprise Information System (EIS) Specialists—Provide domain expertise in the systems that are being integrated using WebLogic Integration adapters. The skills of EIS specialists include technical analysis and application integration design.
- System Administrators—Provide in-depth technical and operational knowledge about databases and applications deployed in an organization. The skills of system administrators include capacity and load analysis, performance analysis and tuning, deployment topologies, and support planning.

Additional Information

To learn more about the software components associated with the adapter, see the following documents:

- *BEA WebLogic Adapter for .NET Release Notes*
<http://edocs.bea.com/wladapters/dotnet/docs812/pdf/relnotes.pdf>
- *BEA WebLogic Adapter for .NET Installation and Configuration Guide*
<http://edocs.bea.com/wladapters/dotnet/docs812/pdf/install.pdf>
- *BEA Application Explorer Installation and Configuration Guide*
<http://edocs.bea.com/wladapters/bae/docs812/pdf/install.pdf>
- *Introduction to the BEA WebLogic Adapters*
<http://edocs.bea.com/wladapters/docs81/pdf/intro.pdf>
- BEA WebLogic Adapters 8.1 Dev2Dev Product Documentation
<http://dev2dev.bea.com/products/wladapters/index.jsp>
- Application Integration documentation
<http://edocs.bea.com/wli/docs81/aiover/index.html>
<http://edocs.bea.com/wli/docs81/aiuser/index.html>
- BEA WebLogic Integration documentation
<http://edocs.bea.com/wli/docs81/index.html>
- BEA WebLogic Platform documentation
<http://edocs.bea.com/platform/docs81/index.html>
- .NET documentation
<http://www.microsoft.com/net>

How to Use This Document

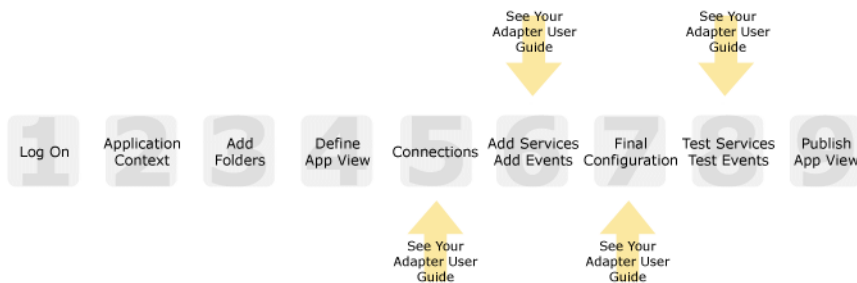
This document is designed to be used in conjunction with *Using the Application Integration Design Console*, available at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

Using the Application Integration Design Console describes, in detail, the process of defining an application view, which is a key part of making an adapter available to process designers and other users. What *Using the Application Integration Design Console* does *not* cover is the specific information about Adapter for .NET that you need to supply to complete the application view definition. You will find that information in this document.

At each point in *Using the Application Integration Design Console* where you need to refer to this document, you will see a note that directs you to a section in your adapter user guide, with a link to the edocs page for adapters. The following roadmap illustration shows where you need to refer from *Using the Application Integration Design Console* to this document.

Figure 1 Information Interlock with *Using the Application Integration Design Console*



Contact Us!

Your feedback on the BEA WebLogic Adapter for .NET 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 .NET documentation.

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

If you have any questions about this version of BEA WebLogic Adapter for .NET, or if you have problems using the BEA WebLogic Adapter for .NET, 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
- 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.
<i>italics</i>	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. <i>Examples:</i> <pre>#include <iostream.h> void main () the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre>
monospace boldface text	Identifies significant words in code. <i>Example:</i> <pre>void commit ()</pre>
<i>monospace italic text</i>	Identifies variables in code. <i>Example:</i> <pre>String <i>expr</i></pre>
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. <i>Examples:</i> <pre>LPT1 SIGNON OR</pre>
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

Convention	Item
[]	<p>Indicates optional items in a syntax line. The brackets themselves should never be typed.</p> <p><i>Example:</i></p> <pre>buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</pre>
	<p>Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.</p>
...	<p>Indicates one of the following in a command line:</p> <ul style="list-style-type: none"> • 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 <p>The ellipsis itself should never be typed.</p> <p><i>Example:</i></p> <pre>buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</pre>
.	<p>Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.</p>

Introducing the BEA WebLogic Adapter for .NET

This section introduces the BEA WebLogic Adapter for .NET and describes how the adapter enables integration with .NET business objects and WebLogic Integration.

It includes the following topics:

- [About the BEA WebLogic Adapter for .NET](#)
- [Getting Started With the Adapter for .NET](#)

About the BEA WebLogic Adapter for .NET

The BEA WebLogic Adapter for .NET connects to your .NET system so that you can easily use your .NET data and functions within your business processes. The adapter provides scalable, reliable, and secure access to your .NET system.

Note: Since .NET is a Windows component, this adapter is supported only on that platform.

This section includes the following topics:

- [About the .NET Framework](#)
- [About the Architecture of the BEA WebLogic Adapter for .NET](#)
- [Supported .NET Operations for Application Integration](#)
- [Supported Services](#)
- [Supported Events](#)
- [Benefits of the Adapter for .NET](#)

About the .NET Framework

The Microsoft .NET Framework is a platform for building, deploying, and running Web Services and applications. It provides a standards-based environment for integrating existing investments with next-generation applications and services as well as the ability to solve the challenges of deployment and operation of Internet-scale applications. The .NET Framework consists of three main parts:

- The common language runtime (CLR), which is the execution engine for .NET Framework applications.
- A hierarchical set of unified class libraries, which includes the Common Language Specification (CLS), is a set of constructs and constraints that serves as a guide for library writers and compiler writers. It enables programmers to use libraries from any language supporting the CLS, and for those languages to integrate with each other. CLS is also important to application developers who are writing code that will be used by other developers. When developers design publicly accessible APIs following the rules of the CLS, those APIs are easily used from all other programming languages that target the common language runtime.
- A componentized version of Active Server Pages called ASP.NET, which is a Web development platform. ASP.NET server controls enable an HTML-like style of declarative programming. Unlike classic ASP, which supports only interpreted VBScript and JScript, ASP.NET supports multiple .NET languages (including built-in support for VB.NET, C#, and JScript.NET).

.NET Assemblies

An assembly is the primary building block of a .NET Framework application. It is a collection of one or more files built, versioned, and deployed as a single implementation unit (as one or more files). All managed types and resources are marked either as accessible only within their implementation unit or as accessible by code outside that unit. Assemblies also play a key role in security. The code access security system uses information about the assembly to determine the set of permissions that code in the assembly is granted.

Assemblies are self-describing by means of their manifest, which is an integral part of every assembly. The manifest:

- Establishes the assembly identity (in the form of a text name), version, culture, and digital signature (if the assembly is to be shared across applications).
- Defines what files (by name and file hash) make up the assembly implementation.

- Specifies the types and resources that make up the assembly, including which are exported from the assembly.
- Itemizes the compile-time dependencies on other assemblies.
- Specifies the set of permissions required for the assembly to run properly.

This information is used at run time to resolve references, enforce version binding policy, and validate the integrity of loaded assemblies. The runtime can determine and locate the assembly for any running object, since every type is loaded in the context of an assembly. Assemblies are also the unit at which code access security permissions are applied. The identity evidence for each assembly is considered separately when determining what permissions to grant the code it contains.

In the .NET context, an executable takes the form of a portable executable (PE) file. The PE can be loaded into memory and executed by the operating system loader. It can be either an .exe or a .dll file. A PE file must be translated by the common language runtime into native code before it can be executed by the operating system.

The file format used for executable programs and for files to be linked together to form executable programs.

Custom Attributes

The common language runtime allows you to add keyword-like descriptive declarations, called attributes, to annotate programming elements such as types, fields, methods, and properties. Attributes are saved with the metadata of a Microsoft .NET Framework file and can be used to describe your code to the runtime or to affect application behavior at run time.

The BEA WebLogic Adapter for .NET uses custom attributes to act as markers to expose methods and classes in your target .NET application and provide the invocation specifications for each exposed method. The BEA Application Explorer generates metadata from the exposed classes and methods to construct service (inbound) schemas.

Microsoft Message Queue

The BEA WebLogic Adapter for .NET also includes support for service (inbound) and event (outbound) adapter integration operations through the use of the Microsoft Message Queue (MSMQ).

Microsoft Message Queuing is a messaging infrastructure and a development tool for creating distributed messaging applications for Microsoft Windows operating systems. The adapter makes

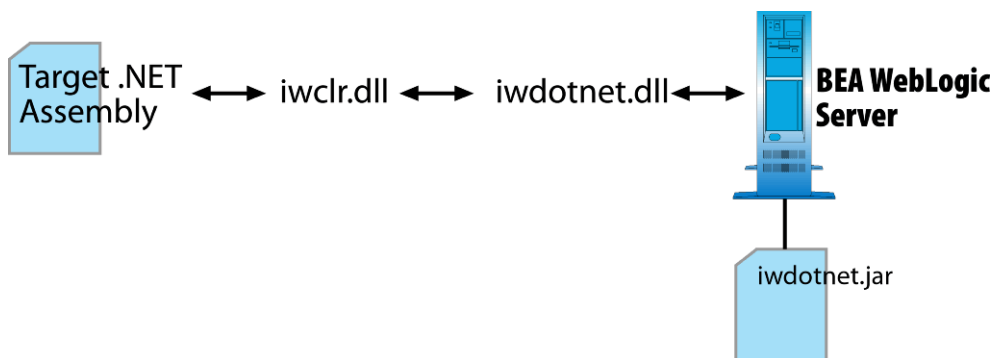
use of the message queue to exchange information with any .NET application. This gives you the flexibility to interact with any .NET application that can interact with a message queue.

When your target .NET application generates messages that arrive on a queue, the adapter detects that as an event. For services, the adapter can place data on a queue for processing by your .NET application.

About the Architecture of the BEA WebLogic Adapter for .NET

The following diagram shows the run-time architecture of the BEA WebLogic Adapter for .NET when performing services that interact directly with your .NET application.

Figure 1-1 Service Run Time

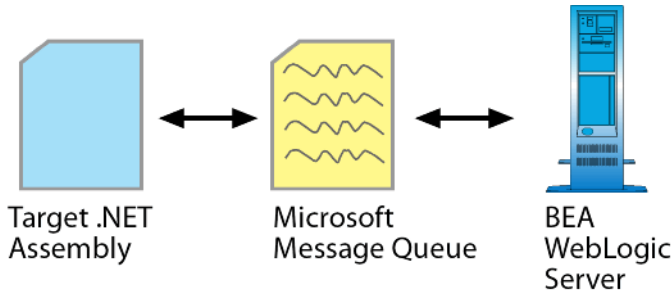


The adapter uses custom DLLs and java classes to ensure seamless integration with your .NET application.

- Target .NET assembly contains the classes and methods that are explored at design time by the BEA Application Explorer or invoked at runtime.
- `iwclr.dll` is a .NET assembly that contains functionality to explore assemblies at design time, load and invoke classes and methods at runtime, and implement the custom attributes used for assembly annotation.
- `iwdotnet.dll` exports the JNI methods required by the Java classes that implement adapter and acts as a common language runtime host.
- `iwdotnet.jar` supplies the classes necessary for adapter implementation in the BEA WebLogic environment.

The following diagram shows the run-time architecture of the BEA WebLogic Adapter for .NET when using MSMQ to integrate with a .NET application. The adapter places and listens for messages on the Microsoft Message queue.

Figure 1-2 Runtime Services and Events Using MSMQ



Supported .NET Operations for Application Integration

The Adapter for .NET supports synchronous and asynchronous, bi-directional message interactions for .NET Business Services, Business Components, and Integration Objects.

It provides integration with the following .NET operations:

- Access to .NET integration objects using XML to handle both services and events
- Access to MSMQ queues for services and events

Supported Services

The Adapter for .NET supports two types of services: DotNet service and MSMQ service. In each case, the adapter sends a file to .NET to cause a .NET business event.

These are the services supported by Adapter for .NET:

- DotNet service
- MSMQ service, which sends a file to a Microsoft Message Queue queue.

Supported Events

The Adapter for .NET supports one type of event: MSMQ event. When an event occurs, the adapter picks up a file from the MSMQ queue and passes it to an event variable within a business process.

These are the events supported by Adapter for .NET.

- MSMQ event, in which the adapter picks up a file from a specific MSMQ queue.

Benefits of the Adapter for .NET

The combination of the adapter and WebLogic Integration supplies everything you need to integrate your workflows and enterprise applications with your .NET system. The Adapter for .NET provides these benefits:

- Integration can be achieved without custom coding.
- Business processes can be started by events generated by .NET.
- Business processes can request and receive data from your .NET system using services.
- Adapter events and services are standards-based. The adapter services and events provide extensions to the *J2EE Connector Architecture (JCA)* version 1.0 from Sun Microsystems, Inc. For more information, see the Sun JCA page at the following URL:

<http://java.sun.com/j2ee/connector/>

- The adapter and WebLogic Integration solution is scalable. The BEA WebLogic Platform provides clustering, load balancing, and resource pooling for a scalable solution. For more information about scalability, see the following URL:

<http://edocs.bea.com/wls/docs81/cluster/index.html>

- The adapter and WebLogic Integration solution benefits from the fault-tolerant features of the BEA WebLogic Platform. For more information about high availability, see the following URL:

<http://edocs.bea.com/wli/docs81/deploy/index.html>

- The adapter and WebLogic Integration solution is secure, using the security features of the BEA WebLogic Platform and the security of your .NET system. For more information about security, see the following URL:

<http://edocs.bea.com/wls/docs81/secintro/index.html>

Getting Started With the Adapter for .NET

This section gives an overview of how to get started using the BEA WebLogic Adapter for .NET within the context of an application integration solution. Integration with .NET involves the following tasks:

- [Step 1: Design the Application Integration Solution](#)
- [Step 2: Determine the Required .NET Business Workflows](#)
- [Step 3: Generate Schemas for .NET Integration Objects](#)
- [Step 4: Define Application Views and Configure Services and Events](#)
- [Step 5: Integrate Your Application With Other BEA Software Components](#)
- [Step 6: Deploy the Solution to the Production Environment](#)

Step 1: Design the Application Integration Solution

The first step is to design an application integration solution, which includes (but is not limited to) such tasks as:

- Defining the overall scope of application integration.
- Determining the business process(es) to integrate.
- Determining which WebLogic Platform components will be involved in the integration, such as web services or workflows designed in WebLogic Workshop, portals created in WebLogic Portal, and so on.
- Determining which external systems and technologies will be involved in the integration, such as .NET systems and other EISs.
- Determining which BEA WebLogic Adapters for WebLogic Integration will be required, such as the BEA WebLogic Adapter for .NET. An application integration solution can involve multiple adapters.

This step involves the expertise of business analysts, system integrators, and EIS specialists (including .NET specialists). Note that an application integration solution can be part of a larger integration solution.

Step 2: Determine the Required .NET Business Workflows

Within the larger context of an application integration project, you must determine which specific .NET integration objects are required to handle services and events to support the business processes in the application integration solution.

Factors to consider include (but are not limited to):

- Type of .NET integration objects, workflows, and transport used to access the .NET system.
- .NET transactions involved in business processes
- Logins required to access .NET transports and perform the required operations
- Whether operations are, from the adapter point of view:
 - services, which notify the .NET system with a request for action, and, in addition, whether such services should be processed synchronously or asynchronously
 - events, which are notifications from the .NET system

This step involves the expertise of .NET specialists, including analysts and administrators.

Step 3: Generate Schemas for .NET Integration Objects

After identifying the .NET integration objects required for the application integration solution, you must generate the XML schemas that will be used to exchange data with one or more .NET systems:

- Services require two XML schemas: one for the .NET request and another for the .NET response.
- Events require a single XML schema to handle the data sent by the .NET system.

You use the BEA Application Explorer tool to generate schemas for .NET operations, or you can generate the schemas yourself. To learn more about schemas, see [Chapter 2, “Generating Schemas for .NET Integration Objects.”](#)

Step 4: Define Application Views and Configure Services and Events

After you create the schemas for your .NET services or events, you create an application view that provides an XML-based interface between WebLogic Server and a particular .NET system within your enterprise. If you are accessing multiple .NET systems, you define a separate application view for each .NET system you want to access. To provide different levels of security access (such as “guest” and “administrator”), define a separate application view for each security level.

Once you define an application view, you can configure events and services in that application view that employ the XML schemas that you created in [“Step 3: Generate Schemas for .NET](#)

Integration Objects” on page 1-8. To learn more about generating schemas, see [Chapter 2, “Generating Schemas for .NET Integration Objects.”](#)

To learn more about defining application views, see [Chapter 3, “Defining Application Views for .NET”](#) in conjunction with *Using the Application Integration Design Console*, at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

Step 5: Integrate Your Application With Other BEA Software Components

Once you have configured and published one or more application views for .NET integration, you can integrate these application views into other BEA software components, such as workflows or Web services created in BEA WebLogic Workshop, or portals built with BEA WebLogic Portal.

For more information, see *Using the Application Integration Design Console*, particularly Chapter 3, “Using Application Views with Application Workflows,” at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

Step 6: Deploy the Solution to the Production Environment

After you have designed, built, and tested your application integration solution, you can deploy it into a production environment. The following list describes some of the tasks involved in deploying an application integration:

- Design the deployment.
- Deploy the required components of the BEA WebLogic Platform.
- Install and deploy the BEA WebLogic Adapter for .NET as described in *BEA WebLogic Adapter for .NET Installation and Configuration Guide*
- Deploy your application views and schemas for .NET integration.
- Verify business processes in the production environment.
- Monitor and tune the deployment.

Generating Schemas for .NET Integration Objects

The Adapter for .NET uses XML documents to communicate with your .NET system's integration objects for both services and events. The format of these XML documents is determined by schemas. For certain services, the format of these XML documents is determined by schemas you generate using the BEA Application Explorer. For other services and for events, you generate schemas manually.

This section explains how to generate schemas. It contains the following topics:

- [Before You Begin](#)
- [About Creating Schemas](#)
- [About the Types of Schemas You Must Generate](#)
- [About the BEA Application Explorer](#)
- [Starting the BEA Application Explorer](#)
- [Setting the Session Path](#)
- [Managing .NET Connections](#)
- [Creating Schemas for Services Using the BEA Application Explorer](#)
- [Creating Schemas for Services and Events Manually](#)
- [Next Steps](#)

Before You Begin

Before you begin to generate schema for the Adapter for .NET, you must:

- Download and install the BEA Application Explorer software. To learn more, see the *BEA Application Explorer Installation and Configuration Guide* at the following URL:

<http://edocs.bea.com/wlapters/docs81/index.html>

- Obtain the information necessary to connect to your .NET system. Contact your .NET administrator for this information.

About Creating Schemas

The BEA WebLogic Adapter for .NET enables you to handle schemas created in two different ways:

- Service schemas created automatically by the BEA Application Explorer

The BEA Application Explorer creates service schemas for services that interact directly with your target .NET application. These service schemas are generated by pointing directly to the assembly directory of your .NET application. To learn more, see “[Creating Schemas for Services Using the BEA Application Explorer](#)” on page 2-10.

- Service and Event Schemas created manually

All events and services in which a message is placed on a queue for consumption by your .NET application require that you manually create the schema. For example, if you are creating a service for a .NET application for which you do not have access to the assemblies, or if you are creating services that interact with a number of .NET applications that pull information from a queue, you enable these services by manually creating the service schemas. To learn more, see “[Creating Schemas for Services and Events Manually](#)” on page 2-12.

About the Types of Schemas You Must Generate

Each service or event the Adapter for .NET uses must be defined by a schema. In order to use services and events, you must generate XML schemas for:

- [Service Requests](#)
- [Service Responses](#)
- [Events](#)

Service Requests

Service requests are requests for action that your application makes to your .NET system. Requests are defined by request schema. As part of the definition, the request schema defines the input parameters required by the .NET system. The .NET system responds to the request with a service response.

Service Responses

Service responses are the way the .NET system responds to a service request. A service response schema defines this service response. Service requests always have corresponding responses.

Events

Events are generated by the .NET system as a result of activity on that system. You can use these events to trigger an action in your application. For example, the .NET system may generate an event when customer information is updated. If your application must do something when this happens, your application is a consumer of this event. Events are defined by event schema.

About the BEA Application Explorer

The BEA Application Explorer uses information in your .NET assembly to generate the schemas required to build application view services. The BEA Application Explorer cannot generate schemas for:

- events
- services with assemblies to which you do not have access
- services that use an MSMQ queue

You must create schemas for these events and services manually. To learn more about creating schemas manually, see [Creating Schemas for Services and Events Manually](#).

This section contains the following topics:

- [About the Process for Defining Schemas Using the BEA Application Explorer](#)
- [Configuring Your .NET Application for Application Explorer Inquiry](#)

About the Process for Defining Schemas Using the BEA Application Explorer

The process for defining XML schemas includes the following steps:

1. [Configuring Your .NET Application for Application Explorer Inquiry](#)
2. [Starting the BEA Application Explorer.](#)
3. [Setting the Session Path.](#)
The BEA Application Explorer uses this path to create the directory for the schemas.
4. [Creating a New Connection](#) or [Using an Existing Connection.](#)
5. [Creating Schemas for Services Using the BEA Application Explorer](#) and [Creating Schemas for Services and Events Manually.](#)

Configuring Your .NET Application for Application Explorer Inquiry

Before you use the BEA Application Explorer to create service schemas, you must configure each target .NET application to enable class and method exploration. The Application Explorer creates service schemas based on the classes and methods you expose in the application. The adapter defines .NET custom attributes that act as markers for which methods are to be exposed and provides the invocation specifications for each exposed method.

Note: You must configure each .NET application with which you want the adapter to exchange data.

1. Locate the assembly for the .NET application for which you must generate metadata.
2. Open the assembly using the Microsoft Visual Studio .NET editor.
3. Import the iwclr.dll file into the assembly.

For example:

```
using System;
using System.Xml;
using System.Text;
using iwclr;
```

4. Revise the code to add the custom attributes, including the location of the method.

Note: All the custom attributes are packaged in iwclr.dll and belong to the iwclr namespace. Adding a reference to iwclr.dll on the local machine makes the attributes available to any .NET project.

For example:

Listing 2-1 Sample DLL Code With Attributes Added

```
[AgentAttribute("Math Agent")]
public class Math
{
    const String ADD_INPUT_SCHEMA = "<xs:schema
xmlns:xs=\"http://www.w3.org/2001/XMLSchema\">" +
        "<xs:element name=\"add\">" +
        "<xs:complexType>" +
        "<xs:sequence>" +
        "<xs:element maxOccurs=\"unbounded\" name=\"parm\"
type=\"xs:int\"/>" +
        "</xs:sequence>" +
        "</xs:complexType>" +
        "</xs:element>" +
        "</xs:schema>";
    const String ADD_OUTPUT_SCHEMA = "<xs:schema
xmlns:xs=\"http://www.w3.org/2001/XMLSchema\">" +
        "<xs:element name=\"total\" type=\"xs:int\"/>" +
        "</xs:schema>";
    public Math()

[ParamsInParamsOutAttribute("Computes the Square Root of a Real Number")]
    public double Sqrt (double number)
    {
        return System.Math.Sqrt(number);
    }

[ParamsInParamsOutAttribute("Computes the sine of a decimal angle in degrees")]
    public double Sine (double angle)
    {
        return System.Math.Sin(angle);
    }
}
```

```

    }

    [ParamsInParamsOutAttribute("Computes the cosine of a decimal angle
in degrees")]
    public double Cosine (double angle)
    {
        return System.Math.Cos(angle);
    }

    [ParamsInParamsOutAttribute("Computes the exponentiation a^b")]
    public double Exponent (double a , double b)
    {
        return System.Math.Pow(a, b);
    }

    [ParamsInParamsOutAttribute("Multiplies two Integers")]
    public int Multiply (int a , int b)
    {
        return a * b;
    }

    [ParamsInParamsOutAttribute("Multiplies two Floats")]
    public float Multiply (float a , float b)
    {
        return a * b;
    }

    [XmlInXmlOutAttribute("Adds one or more integers", "add",
ADD_INPUT_SCHEMA, "total", ADD_OUTPUT_SCHEMA)]
    public XmlElement Add(XmlElement input)

```

Note: For the following descriptions, simple types are any of the .NET primitive types (for example, `System.Int32`, `System.Byte`, etc.) and `System.String`. An XML document by definition is represented using an instance of the .NET `System.Xml.XmlDocument` class.

The attributes are defined as follows:

Table 2-1 Attributes and Their Use

Attribute	Use
AgentAttribute	Applied to classes that must be exposed.
ParamsInParamsOutAttribute	Applied to methods that must be exposed, and have only primitive types or structures or arrays that only use primitive types, as input and output.
XMLInXMLOutAttribute	Applied to methods that must be exposed and have only an XML element as input and an XML element as output.
ParameterAttribute	Applied to give more descriptive information about parameters that are simple types. For example, in a class exposing a divide method, it makes sense to know which of a pair of input parameters of type System.Int32 is the denominator.

5. Save and recompile the assembly.

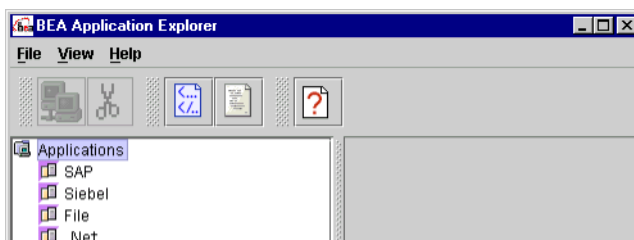
Starting the BEA Application Explorer

You use the BEA Application Explorer to generate service request schemas and service response schemas. The schemas you create are published in the WebLogic Integration repository.

To start the BEA Application Explorer:

1. Open the BEA Application Explorer.
 - In Windows, choose Windows Start→Programs→BEA Application Explorer.

The BEA Application Explorer window appears.



Setting the Session Path

The session path determines the directory where the BEA Application Explorer places your generated XML schemas and connection information. Your schemas are stored here:

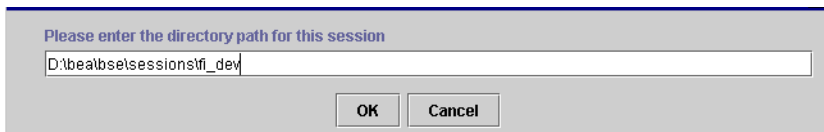
```
session_path\dotnet\connection_name\schemas
```

Here, *connection_name* is the value you specify when you select a connection. To learn more about selecting a connection, see [“Managing .NET Connections.”](#)

To set the session path:

1. From the File menu, choose Session.

The Enter Session Path window appears, displaying a default path.



The screenshot shows a dialog box with a title bar that says "Please enter the directory path for this session". Below the title bar is a text input field containing the path "D:\bea\beasessions\stf_dev". At the bottom of the dialog box, there are two buttons: "OK" and "Cancel".

2. Do one of the following:

- To accept the default session path, click OK.
- To specify a different path, enter the path and click OK.

Specifying a different path allows you to group your schema according to project, or other logical group.

Managing .NET Connections

The BEA Application Explorer must connect to your .NET system before you can generate schemas. Therefore, you must first define a connect to your .NET system.

This section includes the following topics:

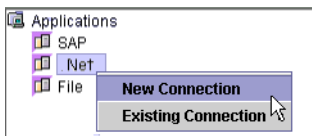
- [Creating a New Connection](#)
- [Using an Existing Connection](#)
- [Disconnecting from .NET](#)
- [Removing Connections](#)

Creating a New Connection

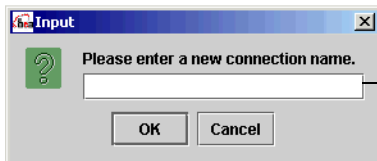
If you are creating a new connection, be sure to check that you have the correct information for your .NET system.

To create a new connection:

1. In the left pane of the BEA Application Explorer window, under Applications right-click .NET→New Connection.



The BEA Application Explorer prompts you for a connection name.



Enter a name for this connection.

2. Enter a name for this connection and click OK.
3. Click OK.

The new connection appears under the `dotnet` node in the BEA Application Explorer window. You can now view business objects and services, as well as all available integration objects in your .NET system.

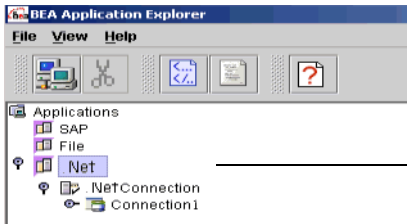
Using an Existing Connection

You can use an existing connection rather than creating a new one.

To use an existing .NET connection:

1. In the left pane of the BEA Application Explorer window, under Applications right-click .NET→Existing Connection→*your connection*.

The connection appears below the .NET node.



Select a connection to use it.

2. If the connection parameters do not correspond to your system, edit them in the .NET Logon Window.
3. Click OK.

Disconnecting from .NET

The BEA Application Explorer allows you to disconnect from .NET.

To disconnect from .NET:

- In the left pane of the BEA Application Explorer, right-click on the connection. Choose Disconnect.

This disconnects from .NET, and the connection icon change to indicate that is not currently connected. To re-establish the connection, right-click on the connection and choose Connect.

Removing Connections

The BEA Application Explorer allows you to remove connections when you no longer need them.

To remove a connection:

- In the left pane of the BEA Application Explorer, right-click on the connection. Choose Remove.

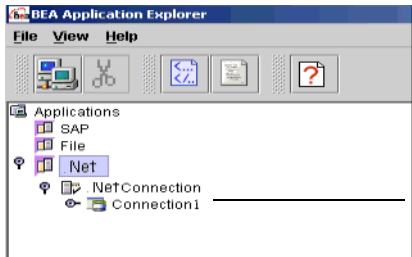
Creating Schemas for Services Using the BEA Application Explorer

Services require two schemas, one for the request and one for the response. Services always have these two schema, even if the response is not used by your application. If your service does not have access to the .NET assembly, you must create the schema manually. Likewise, if your service uses an MSMQ queue, you must create the schemas manually. To learn more about

creating service schemas manually, see [“Creating Schemas for Services and Events Manually” on page 2-12.](#)

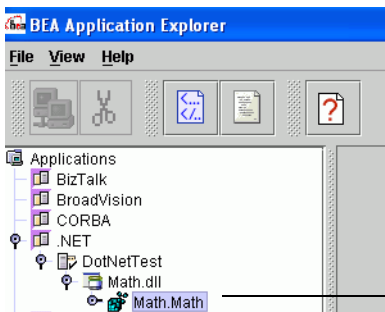
To create a schema for a service:

1. Start BEA Application Explorer. To learn more, see [“Starting the BEA Application Explorer” on page 2-7.](#)
2. Set the session path. This determines where the BEA Application Explorer places your schemas. To learn more, see [“Setting the Session Path” on page 2-8.](#)
3. Select or create a connection to .NET. To learn more, see [“Managing .NET Connections” on page 2-8.](#)
4. Expand the tree under Applications → .NET → *connection name* → Integration Objects to see the items for which you may create a schema. If you cannot expand the tree beneath .NET, you have not set a connection for .NET.



Expand the list of integration objects.

5. Select the method for this schema.

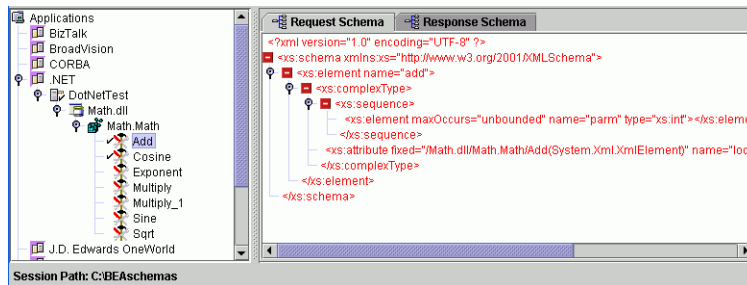


Select a method for this schema.

6. Right-click the item for which you wish to create the schema and choose Create Service Schemas.

Note: For your MSMQ service schemas to work properly, you must do some additional configuration. To learn about this configuration, see [“Configuring Schemas for MSMQ” on page 2-16.](#)

The BEA Application Explorer displays tabs that show the request and response schemas.



The BEA Application Explorer creates a directory structure within the working directory you identified earlier. In this example, the working directory is `C:\BEA\BEASCHEMAS`.

Within this directory, the BEA Application Explorer creates a folder called `DOTNET` as well as subfolders to hold the schemas for each configured `.NET` connection. In this example, the schemas were created in the folder called `DotNetTest`, and the BEA Application Explorer adds the following items to the folder `C:\BEASchemas\DOTNET\DotNetTest`:

`manifest.xml`

`service_Math_dll_Math_Math_Add.xsd`

`service_Math_dll_Math_Math_Add_response.xsd`

You have successfully created service request and response schemas for this `.NET` application.

Creating Schemas for Services and Events Manually

The BEA Application Explorer creates schemas for some `.NET` services. However there are several times when you must create schemas manually:

- You don't have access to the assembly for your `.NET` service
- Your service posts information to an MSMQ queue
- You are using an event. All event schemas must be created manually.

If you use the BEA Application Explorer to create service schemas, use the repository created in that process to store these manually-created schemas. This is the most convenient way to create the repository. If you do not use the BEA Application Explorer to create service schemas, you must create the repository manually.

This topic contains the following subtopics:

- [About Schema Repositories](#)

- [Creating a Schema Repository](#)
- [Creating Schemas](#)

About Schema Repositories

A schema repository consists of the following elements:

- Manifest file (`manifest.xml`) that describes the event and service schemas contained in the repository. For example, `c:\BEASchemas\DotNet\`.
- Event and service schemas. The schemas files have an `xsd` extension.
Schemas describe each event arriving to and propagating out of an adapter. Schemas describe each request sent to and each response received from an adapter. There is one schema for each event, and there are two schemas for each service (one for the request and one for the response).

Creating a Schema Repository

Creating a repository includes the following tasks:

- [Naming Schema Repositories](#)
- [Creating a Manifest.xml File](#)

Naming Schema Repositories

The schema repository is a directory. It has a three-part naming convention. For example, on Windows a repository name is as follows:

session_base_directory\adapter\connection_name

Table 2-2 Schema Repository Naming Convention

Name	Description
<i>session_base_directory</i>	The schema's session base path, which represents a folder under which multiple sessions of schemas may be held.

Name	Description
<i>adapter</i>	The type of adapter (for example, .NET or SAP).
<i>connection_name</i>	The name representing a particular instance of the adapter. For example, WebApp1 can be a connection for a particular .NET application, and WebApp2 can be another; each of these systems having different relevant events and services, and/or security access.

For example, if the session base path is `D:\bea\bse`, the adapter type is `.NET`, and the connection name is `WebApp1`, then the schema repository is the directory:

```
D:\bea\bse\DotNet\WebApp1
```

If you do not already have a repository of the correct name created by the BEA Application Explorer, you must create the directory yourself.

Creating a Manifest.xml File

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. Each repository must have a manifest.

The following is an example of a manifest file showing the relationships between event schemas and service request and response schemas.

Listing 2-2 Sample Manifest.xml File

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<manifest>
<connection>
<directory>D:\DOTNET\DotNet\Assemblies</directory>
<recursive>>false</recursive>
</connection>
  <schemaref name="EXCEL_ADDR_IN">
    <event root="workbook" file="Headers.xsd"/>
    <request root="workbook" file="Headers.xsd"/>
    <response root="emitStatus">
```

```

file="service_MSMQ_response.xsd"/>
    </schemaref>
<schemaref name="Math_dll_Math_Math_Add" alias="Math.dll_Math_Math_Add">
<request root="add" file="service_Math_dll_Math_Math_Add.xsd"/>
<response root="total"
file="service_Math_dll_Math_Math_Add_response.xsd"/>
</schemaref>
<schemaref name="Math_dll_Math_Math_Cosine"
alias="Math.dll_Math_Math_Cosine">
<request root="Cosine" file="service_Math_dll_Math_Math_Cosine.xsd"/>
<response root="CosineResponse"
file="service_Math_dll_Math_Math_Cosine_response.xsd"/>
</schemaref>
</manifest>

```

The manifest has a connection section that specifies the path to the directory where the assemblies for your target application are stored. This can be ignored if you do not use the BEA Application Explorer to generate service schemas. This section is created if you use the BEA Application Explorer to create service schemas for services that connect directly to your .NET application.

The manifest also has a schema reference section, named `schemaref`. The schema reference name appears in the drop-down list on the Add Service or Add Event screens in the WebLogic Integration Application View Console. Each named schema reference can contain three schemas, one of each type.

Creating Schemas

Schemas describe the rules of the XML documents that traverse WebLogic Integration. You can generate a schema manually or by using a schema-generating tool. The following samples demonstrate the kind of schema that you must create for each service that must emit XML documents to a Microsoft Message Queue and for each event that pulls XML documents from a Microsoft Message Queue. An instance document based on that schema follows the following sample schema.

Note: The namespace prefix in the manifest file must be `xsd`.

Configuring Schemas for MSMQ

After you have created your MSMQ schemas, you must do some more configuration for them. This applies to both MSMQ services and MSMQ events.

To create schemas for MSMQ:

1. Create the schemas using the BEA Application Explorer. To learn more about using the BEA Application Explorer, see [“Creating Schemas for Services Using the BEA Application Explorer” on page 2-10](#).
2. In the `BEA_DOTNET_SAMPLES.zip` file included in the adapter distribution, find the following files:

- `Headers.xsd`
- `manifest.xml`
- `service_MSMQ_response.xsd`

To get a copy of the samples, go to the following URL:

http://commerce.bea.com/products/weblogicadapters/wl_adapter_home.jsp

3. Extract these files to the directory where you created the schemas in Step 1. This is typically `session_base_directory\DOTNET`.

This creates a directory named `session_base_directory\DOTNET\msmq` which contains the schemas and manifest file.

Sample Schema File

The following event schema is referenced in the manifest file listed in [“Creating a Manifest.xml File” on page 2-14](#).

Listing 2-3 Sample Schema File

```
<?xml version="1.0" encoding="UTF-8"?>
<!--W3C Schema generated by XML Spy v4.3 U (http://www.xmlspy.com)-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xsd:element name="Sheet1">
    <xsd:complexType>
      <xsd:sequence>
```

```

        <xsd:element ref="row" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="col1" type="xsd:string"/>
<xsd:element name="col2" type="xsd:string"/>
<xsd:element name="col3">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Lord"/>
            <xsd:enumeration value="Mr"/>
            <xsd:enumeration value="Mrs"/>
            <xsd:enumeration value="Sir"/>
            <xsd:enumeration value="TITLE"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="col4" type="xsd:string"/>
<xsd:element name="col5" type="xsd:string"/>
<xsd:element name="col6" type="xsd:string"/>
<xsd:element name="col7">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="COUNTRY"/>
            <xsd:enumeration value="Jamaica"/>
            <xsd:enumeration value="UK"/>
            <xsd:enumeration value="USA"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="row">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="col1"/>
            <xsd:element ref="col2"/>
            <xsd:element ref="col3"/>
            <xsd:element ref="col4"/>
            <xsd:element ref="col5"/>

```

```

        <xsd:element ref="col6" />
        <xsd:element ref="col7" />
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="workbook">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="Sheet1" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>

```

The following listing is an example of an instance document based on the previous schema:

Listing 2-4 Sample XML Instance Document

```

<?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>
    </row>
  </Sheet1>
</workbook>

```



```

        <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>
    <col1>Eileen</col1>
    <col2>Anderson</col2>
    <col3>Mrs</col3>
    <col4>Belle Plain</col4>
    <col5>Maypen</col5>
    <col6>Clarendon</col6>
    <col7>Jamaica</col7>
</row>
</Sheet1>
</workbook>

```

Removing Schemas in the BEA Application Explorer

To remove a schema:

1. Right-click on an integration object for which there is at least one schema.
If there is an event schema defined for this integration object, the menu has a Remove Event Schemas option.
If there are service schemas defined for this integration object, the menu has a Remove Event Schema option.
2. Choose the appropriate option.

Next Steps

After you have defined schemas for your events and services, the next step is to create an application view. An application view makes the services and events available to applications. To learn more about application views, see [Defining Application Views for .NET](#).

Defining Application Views for .NET

An application view is a business-oriented interface to objects and operations within an EIS.

This section presents the following topics:

- [How to Use This Document](#)
- [Before You Begin](#)
- [About Application Views](#)
- [About Defining Application Views](#)
- [Defining Service Connection Parameters](#)
- [Setting Service Properties](#)
- [Setting Event Properties](#)
- [Defining Event Connection Parameters](#)
- [Testing Services](#)
- [Testing Events Using a Service](#)

How to Use This Document

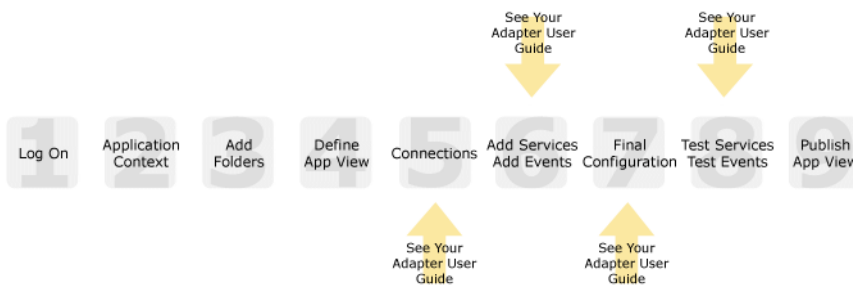
This document is designed to be used in conjunction with *Using the Application Integration Design Console*, available at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

Using the Application Integration Design Console describes, in detail, the process of defining an application view, which is a key part of making an adapter available to process designers and other users. What *Using the Application Integration Design Console* does *not* cover is the specific information—about connections to your .NET system, as well as supported services and events—that you must supply as part of the application view definition. You will find that information in this section.

At each point in *Using the Application Integration Design Console* where you need to refer to this document, you will see a note that directs you to a section in your adapter user guide, with a link to the edocs page for adapters. The following road map illustration shows where you need to refer from *Using the Application Integration Design Console* to this document.

Figure 3-1 Information Interlock with *Using the Application Integration Design Console*



Before You Begin

Before you define an application view, make sure you have:

- Installed and deployed the adapter according to the instructions in *BEA WebLogic Adapter for .NET Installation and Configuration Guide*.
- Determined which business processes need to be supported by the application view. The required business processes determine the types of services and events you include in your application views. Therefore, you must gather information about the application's business requirements from the business analyst. Once you determine the necessary business

processes, you can define and test the appropriate services and events. For more information, see [“Getting Started With the Adapter for .NET” on page 1-6](#).

- Gathered the connection information for your .NET system. To learn more about the connection information needed for your .NET system, see your .NET system administrator.

About Application Views

An application view defines:

- Connection information for the EIS, including login information, connection settings, and so on.
- Service invocations, including the information the EIS requires for this request, as well as the request and response schemas associated with the service.
- Event notifications, including the information the EIS publishes and the event schema for inbound messages.

Typically, an application view is configured for a single business purpose and contains only the services and events required for that purpose. An EIS might have multiple application views, each defined for a different purpose.

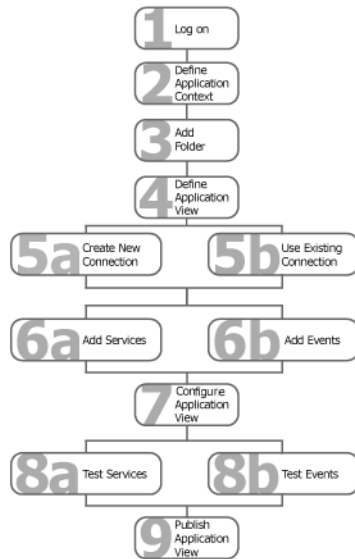
About Defining Application Views

Defining an application view is a multi-step process described in *Using the Application Integration Design Console*, available at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

The information you enter depends on the requirements of your business process and your EIS system configuration. [Figure 3-2](#) summarizes the procedure for defining and configuring an application view.

Figure 3-2 Process for Defining and Configuring an Application View



To define an application view:

1. Log on to the WebLogic Integration Application View Console.
2. Define the application context by selecting an existing application or specifying a new application name and root directory.

This application will be using the events and services you define in your application view. The application view works within the context of this application.
3. Add folders as required to help you organize application views.
4. Define a new application view for your adapter.
5. Add a new connection service or select an existing one.

If you are adding a new connection service, see [“Defining Service Connection Parameters” on page 3-5](#) for details about .NET requirements.
6. Add the events and services for this application view.

See the following sections for details about .NET requirements:

 - [“Setting Service Properties” on page 3-6](#)

- “Setting Event Properties” on page 3-10
7. Perform final configuration tasks.

If you are adding an event connection, see “Defining Event Connection Parameters” on page 3-12 for details about .NET requirements.
 8. Test all services and events to make sure they can properly interact with the target .NET system.

See the following sections for details about .NET requirements:

 - “Testing Services” on page 3-13
 - “Testing Events Using a Service” on page 3-19
 9. Publish the application view to the target WebLogic Workshop application.

This is the application you specified in step 2. Publishing the application view allows workflow developers within the target application to interact with the newly published application view using an Application View control.

Defining Service Connection Parameters



This information applies to “Step 5A, Create a New Browsing Connection” in *Using the Application Integration Design Console*, at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

The Select Browsing Connection page allows you to choose the type of connection factory to associate with the application view. You can select a connection factory within an existing instance of the adapter or create a connection factory within a new adapter instance.

Adapter Instance:

[Create New...](#) _____ Click to create a new connection factory

Existing Adapter Instances:

Adapter Name	Operations	Description

[Back](#) _____ Existing connection factories will be here.

After you enter a connection name and description, you use the Configure Connection Parameters page to specify connection parameters for a connection factory.

To create a new browsing connection:

1. In the Create New Browsing Connections page, enter a connection name and description as described in *Using the Application Integration Design Console*.

The Configure Connection Parameters page appears to allow you to configure the newly created connection factory within the new adapter instance.

On this page, you supply parameters to connect to your EIS

The BEA Application Explorer generates schema information for a session stored at a location that must be known to the general adapter. Enter this session location here. A session can support multiple connections.

Once you have entered the **session path** location, click on the pulldown arrow for the **connection name**, which will display a selection list of valid connections.

Session Path*	<input type="text" value="D:\Program Files\BEA Systems\BEA Application Explorer\sessions"/>	Specify a session path.
Connection Name*	<input type="text" value="IDES"/>	Specify a connection.
<input type="button" value="Connect to EIS"/>		

Note: A red asterisk (*) indicates that a field is required.

2. Specify a session path and connection name.

This information enables the application view to interact with the target .NET system. You need enter this information only once per application view.

3. Click Connect to EIS.

You return to the Create New Browsing Connections, where you can specify connection pool parameters and logging levels. For more information, see *Using the Application Integration Design Console* at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

Setting Service Properties

1 2 3 4 5 **6** 7 8 9

This information applies to “Step 6A, Add a Service to an Application View” in *Using the Application Integration Design Console*, at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

Adapter for .NET uses services to make requests of the .NET system. A service consists of both a request and a response. The Adapter for .NET supports the following services:

- [DotNet Service](#)
- [MSMQ Service](#)

DotNet Service

1 2 3 4 5 **6** 7 8 9

This information applies to “Step 6A, Add a Service to an Application View” in *Using the Application Integration Design Console*, at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

After you create and configure an application view, you can add services that support the application’s functions.

To configure a DotNet Service:

1. Enter a unique service name that describes the function the service performs.
2. Select dotNetService from the Select list.

The Add Services page displays the fields required for this service type.

On this page, you add services to your application view.

Unique Service Name:*

Select: dotNetService

directory*

Note: A red asterisk (*) indicates that a field is required.

3. Enter the following information:

Table 3-1 DotNet Service Parameters

Parameter	Description
directory	Directory where the assemblies for your target .NET application are stored.

4. See “[Common Service and Event Settings](#)” on page 3-9 for information about selecting a schema and configuring tracing.

MSMQ Service

1 2 3 4 5 **6** 7 8 9

This information applies to “Step 6A, Add a Service to an Application View” in *Using the Application Integration Design Console*, at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

After you create and configure an application view, you can add services that support the application’s functions.

To configure an MSMQ Service:

1. Enter a unique service name that describes the function the service performs.
2. Select MSMQEmitter from the Select list.

The Add Services page displays the fields required for this service type.

On this page, you add services to your application view.

Unique Service Name: *

Select: MSMQEmitter ▼

Queue Name*	<input type="text"/>
Correlation ID	<input type="text"/>
Priority	3

Note: A red asterisk (*) indicates that a field is required.

3. Enter the following information:

Table 3-2 MSMQ Service Parameters

Parameter	Description
Queue Name	The name of the Microsoft Message queue on which to post the messages.
Correlation ID	The correlation ID used in the MSMQ message header.
Priority	Sets the priority of the message in the message queue and can be used to determine the order in which messages are retrieved from the queue; 0 is the lowest priority, and 9 is the highest priority.

- See “[Common Service and Event Settings](#)” on page 3-9 for information about selecting a schema and configuring tracing.

Common Service and Event Settings

1 2 3 4 5 **6** 7 8 9

This information applies to “Step 6A, Add a Service to an Application View” in *Using the Application Integration Design Console*, at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

You select a schema and select tracing options the same way for all services.

To set common service settings:

- In the Schema list, select the schema you want to use with this service.

For more information, see [Chapter 2, “Generating Schemas for .NET Integration Objects.”](#)

For MSMQ services and events, use the schema Headers.xsd. To learn more about MSMQ schemas, see “[Configuring Schemas for MSMQ](#)” on page 2-16.

schema:

- Configure logging and tracing for this service, as follows:

Logging captures information from your adapter and writes it in a log file. Tracing displays runtime information in the console. You set the type and amount of information you wish to capture as part of the final configuration tasks. This is described in detail in *Using the Application Integration Design Console*.

settings

Trace on/off	<input type="checkbox"/>
Verbose Trace on/off	<input type="checkbox"/>
Document Trace on/off	<input type="checkbox"/>

- Select the Trace on/off check box to enable tracing for this service. Trace information appears in the runtime console.
 - Select the Verbose Trace on/off check box to enable more detailed tracing for this service.
 - Select the Document Trace on/off check box to enable inclusion of all documents sent or received by the adapter in the trace for this service.
- Click Add to add the service or event.

For more information about the next step, see *Using the Application Integration Design Console* at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

Setting Event Properties

1 2 3 4 5 **6** 7 8 9

This information applies to “Step 6B, Add an Event to an Application View” in *Using the Application Integration Design Console*, at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

An event defines how your application responds to events generated by .NET. The Adapter for .NET supports the following events:

- [MSMQ Event](#)

MSMQ Event

1 2 3 4 5 **6** 7 8 9

This information applies to “Step 6B, Add an Event to an Application View” in *Using the Application Integration Design Console*, at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

In a MSMQ event, the adapter picks up a message from a specific Microsoft Message queue.

To configure an MSMQ Event:

1. Enter a unique event name that describes the function the event performs.

The Add Events page displays the fields required for this event type.

On this page, you add events to your application view.

Unique Event Name: *

MSMQEvent

Queue Name*	<input type="text"/>
Message ID Filter	<input type="text"/>
Correlation ID Filter	<input type="text"/>
Polling Interval	<input type="text"/>
Character Set Encoding*	UTF-8
Error Queue	<input type="text"/>
Error Correlation ID	<input type="text"/>
Error Message Priority	3

Note: A red asterisk (*) indicates that a field is required.

2. Enter the following information:

Table 3-3 MQEvent Parameters

Parameter	Description
Queue Name	The name of the Microsoft Message queue that the adapter for .NET polls for events.
Message ID Filter	Filters messages so that only messages with the ID specified are pulled from the queue.
Correlation ID Filter	The CorrelationId of the message to be received.
Polling Interval	The maximum wait interval (in the format nnH:nnM:nnS) between checks for new documents. The higher this value, the longer the interval, and the fewer system resources that are used. However, with a high value, the worker thread cannot respond to a stop command. If timeout is set to 0, the listener runs once and terminates. Default is 2 seconds.
Character Set Encoding	The character set encoding for inbound documents, for example, UTF-8.
Error Queue	The name of the error queue to be used for dead letters.
Error Correlation ID	The correlation ID used in the MSMQ error message header.
Error Message Priority	Sets the priority of the error message; 0 is the lowest, and 9 is the highest.

3. See [“Common Service and Event Settings” on page 3-9](#) for information about selecting a schema and configuring tracing.

Defining Event Connection Parameters

1 2 3 4 5 6 **7** 8 9

This information applies to “Step 7, Perform Final Configuration Tasks” in *Using the Application Integration Design Console*, at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

Once you have finished adding services and events and have saved your application view, you must perform some final configuration tasks, including configuring event delivery connections, before testing the services and events. You perform these configuration tasks from the Final Configuration and Testing page.

To define event connection parameters:

1. In Connections area on the Application View Administration page, click Select/Edit.
2. In the Event Connection area, click Event to edit the default event connection.

The Configure Event Delivery Parameters page appears.

On this page, you supply parameters to configure event delivery for this Application View

Password:	<input type="text"/>	Enter connection information for your system.
SleepCount:	<input type="text"/>	
UserName:	<input type="text"/>	
<input type="button" value="Continue"/>		

Note: A red asterisk (*) indicates that a field is required.

3. Enter the following information:

Table 3-4 Event Connection Parameters

Parameter	Description
username	Your WebLogic Server Administration Console user name, defined in the startWebLogic script
password	The password for your WebLogic Server Administration Console user name
SleepCount	The number of seconds the adapter will wait between polling for events

The event delivery parameters you enter on this page enable connection to your .NET system and are used when generating events. The parameters are specific to the associated adapter and are defined in the `wli-ra.xml` file within the base adapter.

- Click **Save** to save your event delivery parameter settings. Click **Continue** to return to the Edit Event Adapter page, and then click **Back** to return to the Final Configuration and Testing page.

The Edit Event Adapter page allows you to define event parameters and configure the information that will be logged for the connection factory. Select one of the following settings for the log:

- Log errors and audit messages
- Log warnings, errors, and audit messages
- Log informational, warning, error, and audit messages
- Log all messages

Note: For maximum tracing, select **Log all Messages**. This is the recommended setting to use when you are collecting debugging information for BEA support.

The table that follows describes the type of information that each logging message contains.

Table 3-5 Logging message categories

This type of message	Contains
Audit	Extremely important information related to the business processing performed by an adapter.
Error	Information about an error that has occurred in the adapter, which may affect system stability.
Warning	Information about a suspicious situation that has occurred. Although this is not an error, it could have an impact on adapter operation.
Information	Information about normal adapter operations.

Testing Services



This information applies to “Step 8A, Test an Application View’s Services” in *Using the Application Integration Design Console*, at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/index.html>

The purpose of testing an application view service is to evaluate whether that service interacts properly with the target .NET system. When you test a service, you supply any inputs required to start the service. For the Adapter for .NET, the input is in the form of a valid XML string that acts as input for the service.

You can test both types of .NET services:

- [Testing a dotNetService](#)

To learn more about testing a DotNet service, see “[Testing a dotNetService](#)” on page 3-14.

- [Testing an MSMQ Service](#)

To learn more about testing a MSMQ service, see “[Testing an MSMQ Service](#)” on page 3-16.

Note: You can test an application view only if it is deployed and only if it contains at least one event or service.

Testing a dotNetService

To test a DotNet service that interacts directly with a target .NET application:

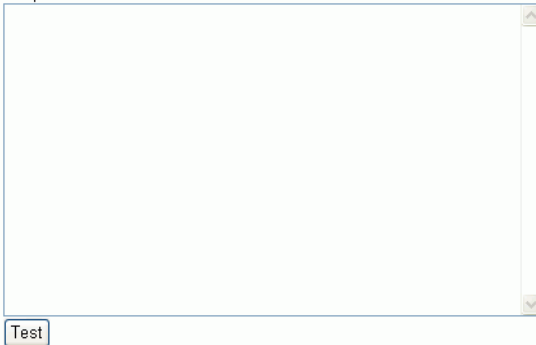
1. In the Summary for Application View page, click the Test link beside the service to be tested.

The Test Services page opens.

Please fill in any inputs to the service query and click Test.

Test Service: TestService on application view 'DotNetTest'

Use the text box below to enter a valid XML string to act as the request data to be sent in this service invocation.



2. Enter the sample request document that matches the request schema for the selected service, for example,

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSPY v5 rel. 4 U
(http://www.xmlspy.com)-->
<add xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\BEAschemas\DOTNET\DotNetTest\service_
Math_dll_Math_Math_Add.xsd"
location="/Math.dll/Math.Math/Add(System.Xml.XmlElement)">
<parm>1</parm>
<parm>2</parm>
<parm>3</parm>
</add>
```

3. Click Test.

The Test Results page displays the XML Response document generated after a successful execution.

This page shows the results from testing a service.

Input to service TestService on application view DotNetTest

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSPY v5 rel.
4 U (http://www.xmlspy.com)-->
<add xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="C:\BEAschemas\DOTN
ET\DotNetTest\service_Math_dll_Math_Math_Add.xsd"
location="/Math.dll/Math.Math/Add
(System.Xml.XmlElement) ">
  <parm>1</parm>
  <parm>2</parm>
  <parm>3</parm>
</add>
```

Output from service TestService on application view DotNetTest

```
<?xml version="1.0"?>
<total>6</total>
```

Testing an MSMQ Service

To test an MSMQ Service:

1. In the Summary for Application View page, click the Test link beside the service to be tested.

The Test Services page opens.

Please fill in any inputs to the service query and click Test.

Test Service: MSMQTest on application view 'DotNetTest'

Use the text box below to enter a valid XML string to act as the request data to be sent in this service invocation.

The screenshot shows a web interface for testing a service. It features a large, empty text input area with a vertical scrollbar on the right side. Below the input area is a small button labeled 'Test'.

2. Enter the sample request document that matches the request schema for the selected service, for example,

```
<?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>ST
ATE</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>Springfield</col5><col6>New
York</col6><col7>USA</col7>

</row><row><col1>Jonathon</col1><col2>Platt</col2><col3>Mr</col3><col4>Wem
bley Point, Harrow
Road</col4><col5>London</col5><col6>MIDDX</col6><col7>UK</col7>

</row><row><col1>Eileen</col1><col2>Anderson</col2><col3>Mrs</col3><col4>B
elle
Plain</col4><col5>Maypen</col5><col6>Clarendon</col6><col7>Jamaica</col7><
/row>
```

```
</Sheet1></workbook>
```

3. Click Test.

The results appear in the WebLogic test window.

The Test Results page displays the XML Response document generated after a successful execution.

This page shows the results from testing a service.

Input to service MSMQTest on application view DotNetTest

```
<?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><c
ol3>Mr</col3><col4>2 Penn Plaza</col4><col5>New
York</col5><col6>New York</col6><col7>USA</col7>

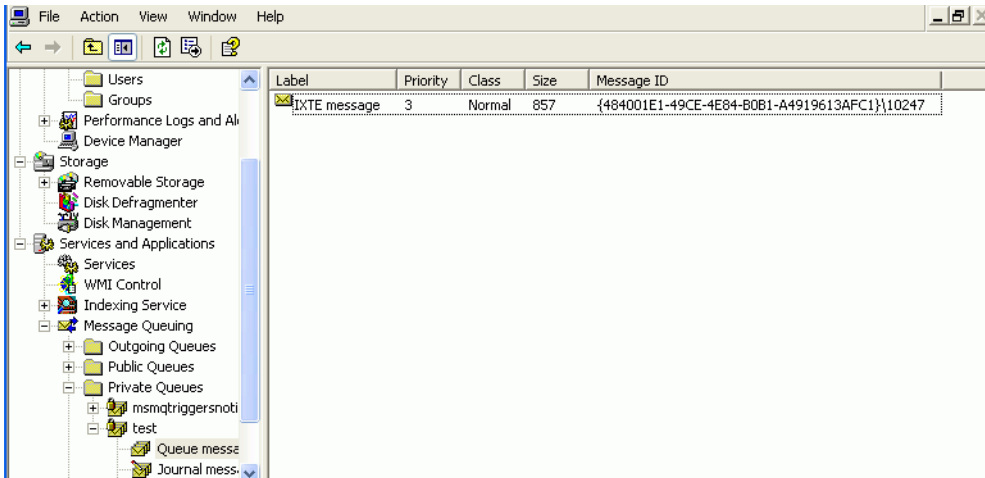
</row><row><col1>Jeff</col1><col2>Paoletti</col2>
<col3>Sir</col3><col4>2 Penn
Plaza</col4><col5>New York</col5><col6>New
York</col6><col7>USA</col7>
```

Output from service MSMQTest on application view DotNetTest

```
<?xml version="1.0"?>
<emitStatus>
  <protocol>MSMQ</protocol>
  <parms>queue=barbarell\private$\test,
priority=3, correlid=null</parms>
  <status>0</status>
  <native>0</native>
  <msg/>
  <msgid>base64(4QFA5M5JhE6wsaSRlhOvwQMgAAA=)
</msgid>
  <correlid>base64(AA=)</correlid>
  <timestamp>2003-11-12T18:49:23.391Z</timestamp>
  <attempts>1</attempts>
</emitStatus>
```

Execution time: 420 (ms)

The message appears on the Microsoft Message Queue supplied during the service configuration. Check the queue for the message.



Testing Events Using a Service

This information applies to “Step 8B, Test an Application View’s Events” in Defining an Application View in Using the Application Integration Design Console, at the following URL:

<http://edocs.bea.com/wli/docs81/aiuser/2usrdef.html>

The purpose of testing an application view event is to ensure that the adapter correctly handles events generated by your target .NET application. When you test an event, you can trigger the event using a service or manually.

Note: You can test an application view only if it is deployed and only if it contains at least one event or service. To learn more about deploying an application view, see *Deploying WebLogic Integration Solutions*.

To test an MSMQ Event using a service:

1. In the Application View Administration page, click the Test link beside the event to be tested. The Test Events page opens.

This page allows you to test an event. You may create the event by invoking a service, or by manually creating the event.

If you want to use a service invocation to create an event, select the service option below, and select the service to invoke. Optionally, you can create the event manually using any tools your EIS provides (for example an interactive SQL tool for the DBMS adapter used to insert a new row to create an insert event).

How do you want to create the event?

Service

Manual

How long should we wait to receive the event?

Time (in milliseconds):

2. Click **Service** and select a service that triggers the event you are testing.
3. In the **Time** field, enter a reasonable period of time to wait, specified in milliseconds, before the test times out (One second = 1000 milliseconds. One minute = 60,000 milliseconds.).
4. Click **Test** and enter the XML string needed to trigger the service.

The service is executed.

Summary

This page shows the results from testing an event.

Generated event of type TestEvent on application view DotNetTest

```
<?xml version="1.0"?>
<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>
    </row>
  </Sheet1>
</workbook>
```

Input to service MSMQTest on application view DotNetTest

```
<?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><c
ol3>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>
```

Output from service MSMQTest on application view DotNetTest

```
<?xml version="1.0"?>
<emitStatus>
  <protocol>MSMQ</protocol>
  <parms>queue=barbarell\private\test,
priority=3, correlid=null</parms>
  <status>0</status>
  <native>0</native>
  <msg/>
  <msgid>base64(4QF&SM5JhE6wsaSR1hOvwQEg&AA=)
</msgid>
  <correlid>base64(AA=) </correlid>
  <timestamp>2003-11-12T17:43:11:46&Z</timestamp>
  <attempts>1</attempts>
```

- If the test succeeds, the Test Result page opens. It shows the event document, the service input document, and the service output document.
- If the test fails, the Test Result page displays only a Timed Out message.

Index

Symbols

- .NET Assemblies
 - described 1-2
- .NET framework
 - described 1-2

A

- adapter
 - benefits 1-6
- Application Explorer
 - about the BEA Application Explorer 2-3
 - starting 2-7
- application views
 - adding events to 3-10
 - adding services to 3-6
 - events, adding 3-10
 - final configuration tasks 3-12
 - overview of defining 3-3
 - preparing to define 3-2
 - services, adding 3-6
 - services, testing 3-13
 - testing services 3-13
- ASP.NET
 - described 1-2
- auditing events 3-13

B

- BEA WebLogic Adapter for .NET, overview of 1-1
- benefits of adapter 1-6

C

- common language runtime (CLR)
 - described 1-2
- Common Language Specification (CLS)
 - described 1-2
- connections
 - creating a new connection 2-9
 - editing an existing connection 2-9
- Custom Attributes
 - described 1-3
- customer support contact information ix

D

- default session path, changing 2-8
- defining schemas 2-4
- DotNet service
 - configuring 3-7

E

- events
 - about 2-3
 - adding to application views 3-10
 - auditing 3-13

G

- getting started 1-6

L

- logging 3-13

M

- Microsoft Message Queue
 - described 1-3
- MSMQ event
 - configuring 3-10

P

product support ix

R

related information viii

S

schemas

- defining 2-4

- events 2-3

- service requests 2-3

- service responses 2-3

service requests 2-3

service responses 2-3

services

- adding to application views 3-6

- testing 3-13

session path, changing 2-8

support ix

supported operations 1-5

T

technical support ix