

Oracle® Agile

Engineering Data Management

Enterprise Integration Platform Development Guide for Agile e6.1.3.0

Part No. E50950-01

January 2014

Copyright and Trademarks

Copyright © 1995, 2014,Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

This documentation is in preproduction status and is intended for demonstration and preliminary use only. It may not be specific to the hardware on which you are using the software. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to this documentation and will not be responsible for any loss, costs, or damages incurred due to the use of this documentation.

The information contained in this document is for informational sharing purposes only and should be considered in your capacity as a customer advisory board member or pursuant to your beta trial agreement only. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle Software License and Service Agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

CONTENTS

Copyright and Trademarks iii
Preface vi
Introduction 1
Connectors 1
Web Services 1
Connectors 3
Configuration 3
Implementation 4
Connector Types 4
Connector Modes 5
Connector Methods 6
Thread Safety 9
WebServices 11
Implementation 11
Configuration 11
Deployment 12
Additional Documentation 13

Preface

The Oracle documentation set includes Adobe® Acrobat™ PDF files. The Oracle Technology Network (OTN) Web site (http://www.oracle.com/technology/documentation/agile.html) contains the latest versions of the Oracle Agile EDM PDF files. You can view or download these manuals from the Web site, or you can ask your Agile administrator if there is an Oracle Documentation folder available on your network from which you can access the documentation (PDF) files.

Note To read the PDF files, you must use the free Adobe Acrobat Reader™ version 7.0 or later. This program can be downloaded from the Adobe Web site (http://www.adobe.com).

Note Before calling Agile Support about a problem with an Oracle Agile EDM manual, please have the full part number, which is located on the title page.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, 7 days a week. For TTY support, call 800.446.2398. Outside the United States, call +1.407.458.2479.

Readme

Any last-minute information about Oracle Agile EDM can be found in the Release Notes file on the Oracle Technology Network (OTN) Web site (http://www.oracle.com/technology/documentation/agile.html)

Agile Training Aids

Go to the Oracle University Web page (http://www.oracle.com/education/chooser/selectcountry_new.html) for more information on Agile Training offerings.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Chapter 1

Introduction

This manual should enable software developers both to implement own connectors for the Enterprise Integration Platform and to implement own web services for the WebServiceConnector.

Connectors

Implementing an additional connector consists of the following steps:

- Provide the application specific configuration parameters in the configuration XML file, e.g. how to connect to the application (login, password, etc.).
- Develop the connector itself, which will be called by the Enterprise Integration Platform controller (kernel) e.g. based on events. The connector either receives data from the Controller in order to send data (or to execute a function) in the external application or reads data from the external application and sends it to the controller for further processing.

Web Services

Implementing an additional Web Service consists of the following steps:

- Provide the web service specific configuration parameters in the configuration XML file, e.g. how your web service can be called from a web service client.
- Develop the web service itself that allows web service calls to be made.
- Provide web service deployment information to get it properly deployed inside the EIP's web server.

Chapter 2

Connectors

Configuration

The configuration file is based on XML. Upon the startup of the Enterprise Integration Platform, the controller reads the configuration file in order to know which connectors need to be started etc.

The configuration data is converted into an internal XML Data Object (XDO), which the controller provides to each connector. The connector itself is responsible for pulling the connector-specific information out of the configuration XDO.

Below is an example of the minimum configuration for a connector. The tags “name”, “version”, “class” and “active” are required.

```
<connector name="example" version="2.2.0" active="false"
class="com.eigner.eai.connector.ExampleConnector">
</connector>
```

In detail, adding a connector would require to add a section for this specific connector. It depends on the functionality of the connector, what information needs to be provided e.g. connection parameters or available functions of the interface. It is recommended to put all connection related parameters under a connection tag that can be accessed easily from within the connector’s source code.

```
<connector name="plm" version="2.2.0" active="true"
class="com.eigner.eai.connector.plm.PlmConnector">
      …
      <connection name="default" active="true">
            <host>plm_server</host>
            <socket>16067</socket>
            <env>axalantORIGIN</env>
            <user>EDB-EIP</user>
            <pwd>{PLM-AES-
128}+YEVllbJyH1/jvIH9OgsFY21pmz8LbcuR2OWYJ7Uzog=</pwd>
            <id>''</id>
            <connection-timeout>300000</connection-timeout>
            <call-timeout>300000</call-timeout>
      </connection>
      …
      </connector>
```

The connector also needs to be defined either as source or target connector in the workflow area. This describes the initial direction of the data transfer:

```
<workflow name="plm-erp" active="true" type="asynchronous">
     <source>plm</source>
     <target>erp</target>
     <request-pipe>plm-erp</request-pipe>
     <response-pipe>erp-plm</response-pipe>
</workflow>
```

The name and path of the mapping file (XSL) must also be provided. The mapping file is responsible for converting the XDO data from the source system format to the target system format.

```
<pipe name="plm-erp">
     <path>${eai.conf}/plm_erp.xsl</path>
</pipe>
```

Implementation

The connector itself is a Java class, which must provide (extend) certain methods like init, warmup, start, stop, and release from the connector interface (see JavaDoc for package com.eigner.eai.connector).

Connector Types

There are two types of connectors: synchronous and asynchronous. The most common one is the asynchronous connector.

To determine which type of connector to develop, the modes of operation are important.

- Asynchronous connector

  Used to gather data from its system if it is the source connector. The data is stored into an internal queue inside the EIP by the controller and the source connector is done for now and waits for the next transfer order. The controller then reads the data from this queue and sends them to the target connector. The target connector processes the data and sends them back to the controller, which stores them into the queue again. The data is then read by controller and sent back to the source connector.

- Synchronous connector

  Used to gather data from its system, sends them in synchronous mode through the controller to the target connector, and waits for the results to send them to its system.

Depending on its purpose the connector must implement either the AsyncConnector interface (methods sendToController and receiveFromController) or the SyncConnector interface (method process). For convenience reasons, there are the abstract class AbstractConnector, AbstractAsyncConnector, and AbstractSyncConnector (also available from the package com.eigner.eai.connector).

Asynchronous Connector

An asynchronous connector usually reads the data from its system when the controller is requesting them (sendToController). This data is stored in an EIP internal queue, and the connector is able to process further requests. When results are returned, the controller triggers the connector by calling its receiveFromController method.

An asynchronous connector must inherit from the com.eigner.eai.connector.AsyncConnector interface. There is also an abstract base class AbstractAsyncConnector, which is recommended to be used since it already implements most of the basic methods.

Note For an example please refer to the file ExampleAsyncConnector.java in the docs directory.

Synchronous Connector

A synchronous connector is mostly triggered by its system, usually reads the needed data, sends them to the controller (via the connector's "process" method) and waits for the results.

A synchronous connector must inherit from the com.eigner.connector.SyncConnector interface. There is also an abstract base class AbstractSyncConnector, which is recommended to be used since it already implements most of the basic methods.

Note For an example please refer to the file ExampleSyncConnector.java in the docs directory.

Connector Data

The vehicle to transfer the data from controller to connector and back is a BusinessObject (BO) or a part of the BO (see JavaDoc for package com.eigner.commons.businessobject). The BO is based on a XDO (XML Data Object, see JavaDoc for package com.eigner.commons.dataobject), and consists of a control area (that is maintained by the controller) and a data area that holds the data for the connector. For further information on the structure of a BO, please refer to the Enterprise Integration Platform Administration Guide for Agile e6.1.3.0.

Connector Modes

A connector should support at least one of the two possible modes: source mode (MODE_SOURCE) or target mode (MODE_TARGET).

Source mode means that the connector can act as a data source. It may then be used as a source in a workflow definition or in a receive activity inside a BPM process.

Target mode means that the connector can act as a data target. It may then be used as a target in a workflow definition or in an "invoke" or "reply" activity inside a BPM process.

As a convenience, there is also a mode MODE_BOTH for connectors that support both.

The mode has to be returned by the connector's getMode() method.

As a conclusion, a connector that supports MODE_SOURCE has to implement the sendToController() method, whereas a connector that supports MODE_TARGET has to implement the receiveFromController() method.

Connector Methods

We will discuss the single connector methods now in more detail.

It is assumed that there is already a custom connector class that has been derived properly from one of the abstract base classes.

The constructor and the init method will only be called if the connector's configuration is set active in the eai_ini.xml.

Constructor

```
public ExampleSyncConnector()
```

The constructor must be a default constructor (without any arguments). It must call the super constructor with the connector's class name, which will be used by the logging framework. You will have access to the logger inside the connector's source code by calling the method getLogger (). Please refer to the JavaDoc of the package com.eigner.commons.logging for further information.

It must then call the setVersion () method with a version number string (e.g. "2.2.0") that must be the same as the one defined in the connector's configuration inside the eai_ini.xml. It should be the same as the current EIP version. The controller uses these version numbers to determine if a configuration is compliant with the connector.

The version numbering consists of three numbers where the first one is the major version, the second one is the minor version and the last one is the revision version. It is required that a connector that differs only by the revision number should read its configuration of a lesser or equal revision number (e.g. a connector with version 2.2.2 should read a configuration for version 2.2.0). This is not required if the versions differ on the major or minor version.

init()

```
public void init(ControllerInstance controller, String connectorName)
```

The controller calls this method after the constructor is called. There the connector's class members should be initialized as well as third party APIs (if any is needed to communicate with the external system).

It is required to call the super init method before all other code. Then you should read in the

connector's connection configuration by calling the controller's getConnectionContext() method. For further processing of the returned element object, please refer to the JavaDoc of package com.eigner.commons.dataobject.

Each connector may have a BOR (Business Object Repository) assigned if needed that defines the calls into the external system depending on the direction (e.g. "SEND"), the business object (e.g. "BOM") and a verb (e.g. "RELEASE"). Since this BOR is highly dependent on the external system, no general advice can be given here. The BOR can be accessed inside the source code by calling the controller's getBorContext() method. The further processing of the returned element object is equivalent to the one returned by getConnectionContext()).

Then you may call the third party's API as needed to initialize it.

warmup()

```
public void warmup() throws ConnectorException
```

The controller calls this method after method init() and before method start(). This allows you to do further initialization that depends on a fully initialized connector before it is started. It is mostly sufficient to not overwrite this method but to use the base class' default implementation.

start()

```
public void start() throws ConnectorException
```

The controller calls this method when the connector should connect to its system. This is only done when the dynamic-connect feature is not activated or the connector is a source connector as defined in the workflows.

The dynamic-connect feature is an optional configuration tag for the connector inside the eai_ini.xml. For further information, please refer to the Enterprise Integration Platform Administration Guide for Agile e6.1.3.0.

sendToController()

```
public boolean sendToController() throws ConnectorException,
UnavailableException
```

The controller calls this method periodically if the connector is derived from AsyncConnector. The interval is defined in the controller's configuration via the parameter polling-interval inside the eai_ini.xml.

This method should read data from the external system, construct a DataArea with these data, and send it to the controller. If a connection problem occurs when reading the data from the system, a UnavailableException should be thrown. If another error occurs, a ConnectorException should be thrown.

The method should return the value "true", if the controller's send() method had been called. Otherwise, the value "false" should be returned.

receiveFromController()

```
public boolean receiveFromController(BusinessObject bo) throws
ConnectorException, UnavailableException
```

The controller calls this method when the controller has data that should be delivered to the connector in asynchronous mode. The connector may query the BO's control area, to determine the type of the BO (e.g. isResponse() for data sent by another connector or previously sent data via its own sendToController() method), or an error state (by calling hasError()).

This method may write data to the external system, construct a ReturnArea with this status information, and send it to the controller. If a connection problem occurs when writing the data to the system, an UnavailableException should be thrown. If another error occurs, a ConnectorException should be thrown.

The method should return the value "true", if the controller's send() method had been called. Otherwise, the value "false" should be returned.

process()

```
public BusinessObject process(String id) throws ConnectorException,
UnavailableException
public BusinessObject process(Collection params) throws
ConnectorException, UnavailableException
```

The external system may call these methods when it wants to have data transferred in synchronous mode. Depending on the third party API, you may also have your own class that calls directly the controller's process() method.

The first method takes a GUID string that is used to identify and read the data from the external system.

The second method gets all the required parameters that are needed to process the synchronous request. The connector may then write the data into the external system, or just use them to construct a DataArea and call the controller's process method. If a connection problem occurs when writing the data to the system, an UnavailableException should be thrown. If another error occurs, a ConnectorException should be thrown.

The method should return the BO that is received as a return value from the controller's process() method.

snapshot()

```
public String snapshot(Collection params) throws ConnectorException,
UnavailableException
```

The external system may call this method when it wants to have data read and stored in a snapshot field in synchronous mode. This method should not make any calls into the controller.

When running in asynchronous mode, the EIP reads the data that should be transferred not at the moment when the request is done. Depending on the polling-interval, the current amount of transfers and the availability of connectors, this is a point of time in the near or far future. To have the data collected at the moment the request is made, the snapshot() method is intended to be used. The data should be stored in the external system to be accessed later in asynchronous mode.

The method should return the GUID of the data that it has read from the external system.

stop()

```
public void stop() throws ConnectorException
```

The controller calls this method when the connector should disconnect from its system. This is done when the dynamic-connect feature is activated and the transfer has been completed or the EIP is terminating.

The dynamic-connect feature is an optional configuration tag for the connector inside the eai_ini.xml. For further information, please refer to the Enterprise Integration Platform Administration Guide for Agile e6.1.3.0.

release()

```
public void release() throws ConnectorException
```

The controller calls this method when the connector is terminated. This is usually only the case when the EIP terminates. This method is the counterpart to the method init().

getMode()

```
public int getMode()
```

This method returns the connector mode (MODE_SOURCE, MODE_TARGET, or MODE_BOTH).

Thread Safety

When implementing the connector please ensure that the access to member variables is thread-safe when they are used in multiple methods.

Please have also in mind to use the smallest synchronization blocks as possible. You should not synchronize the interface methods since this may lead to locking problems on the connector itself.

Chapter 3

WebServices

To provide own Web Services to the Enterprise Integration Platform, you have to create an implementation class, add it to the configuration and deploy it to the Web Server's directory.

Implementation

When implementing the Web Service Java class, you may inherit your class from com.eigner.eai.connector.net.ws.WebService. This abstract base class provides you with a Logger instance and convenience methods for handling the Web Service request.

Basically, your code should do the following:

```
// The context (name of the web service connector) must be either
provided by
// the method call or by a dynamic mapping.
WebServiceContext wsc = WebServiceContextFactory.getContext(context);

// Extract the required data from the method call.
// key: Unique key for data
// noun: Business Object noun (e.g. ITEM)
// verb: Business Object verb (e.g. CREATE)
// message: XML data
// language: Language code (see Common Section in the Administration
Manual)
// synchronous: Flag for synchronous data transmission
String resultString = wsc.process(key, noun, verb, message, language,
synchronous);

// Prepare the result XML string for the return value of the method
call (if any)
```

For further information regarding the data, please refer to the chapter about the XDOs in the Enterprise Integration Platform Administration Guide for Agile e6.1.3.0.

Configuration

Depending on the type of connector, the definition of the Web Service must be added to the synchronous connector's configuration or the asynchronous connector's configuration, although the definition is identical.

Example:

```
<connector name="ws" version="2.2.0" active="false"
class="com.eigner.eai.connector.net.WebServiceConnector">
      <connection name="default" active="true">
            …
            <service name="myservice" wsdd="/com/foo/MyService.wsdd"
location="/axis/services/MyService"/>
            …
      </connection>
      …
</connector>
```

Deployment

For deploying your Web Service into the EIP, please provide a proper WSDD (Web Service Deployment Description) file.

Example:

```
<deployment name="eip" xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
      <service name="MyService" provider="java:MSG">
            <parameter name="className" value="com.foo.MyService"/>
            <parameter name="allowedMethods" value="*"/>
      </service>
</deployment>
```

For further information, please see http://ws.apache.org/axis/java/user-guide.html (esp. “Custom Deployment - Introducing WSDD” and “Service Styles - RPC, Document, Wrapped, and Message”).

Then pack your Java classes and this WSDD file into a JAR and deploy it to the webapps’s axis/WEB-INF/lib directory. The webapps’s directory is specified in the file conf/eai_ini.xml under eai-root/controller/webserver.

You may need to delete the file axis/WEB-INF/server-config.xml and restart the Integration Platform in order to use your Web Service.

Chapter 4

Additional Documentation

In addition to this document, Agile Software also provides the Java Documentation of Java classes and methods, which are necessary to develop your connector.

The Javadoc package (HTML files) includes following pages (which are located in docs/apidocs directory):

- Documentation of the BusinessObject class and its relatives (com.eigner.commons.businessobject) that represent the internal data structure which is sent and received by the connectors.
- Documentation of the XDO and XDOTransformer classes (com.eigner.commons.dataobject), which are necessary for the creation and parsing of XML Data Objects.
- Documentation of the connector interfaces (com.eigner.eai.connector), which explains what methods a connector needs to implement and which exceptions (ConnectorException and UnavailableException) can be thrown into the application controller.
- Documentation of the ControllerInstance interface (com.eigner.eai.connector) that is used to interact with the controller, of the ContextException class (com.eigner.commons.config) that may be thrown, and of the Decrypter interface (com.eigner.commons.crypt) that may be used to decrypt sensible data like passwords.
- Documentation of the logging framework.
- Documentation of the NestedException and NestedRuntimeException classes (com.eigner.commons.lang) which are base classes for most of the actually thrown exceptions.
- Documentation of some utility classes, which you may use when needed.