Oracle® Agile Product Lifecycle Management for Process

Content Synchronization and Syndication Configuration Guide

Release 6.1.0.3

E39986-01

March 2013

ORACLE

Oracle Agile Product Lifecycle Management for Process Content Synchronization and Syndication Configuration Guide, Release 6.1.0.3

E39986-01

Contents

Preface

The *Agile Product Lifecycle Management for Process Content Synchronization and Syndication Configuration Guide* discusses basic configuration information for the Content Synchronization and Syndication application of Oracle Agile Product Lifecycle Management (PLM) for Process.

This Preface contains these topics:

Audience

This guide is intended for end users who are responsible for creating and managing information in Agile PLM for Process. Information about administering the system resides in the *Agile Product Lifecycle Management for Process Administrator User Guide*.

Variability of Installations

Descriptions and illustrations of the Agile PLM for Process user interface included in this manual may not match your installation. The user interface of Agile PLM for Process applications and the features included can vary greatly depending on such variables as:

- Which applications your organization has purchased and installed
- Configuration settings that may turn features off or on
- Customization specific to your organization
- Security settings as they apply to the system and your user account

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following documents in the Agile PLM for Process Release 6.1 documentation set:

- *Agile Product Lifecycle Management for Process Administrator User Guide*
- *Agile Product Lifecycle Management for Process Configuration Guide*
- *Agile Product Lifecycle Management for Process Content Synchronization and Syndication User Guide*
- Agile *Product Lifecycle Management for Process Security Configuration Guide*
- *Agile Product Lifecycle Management for Process Release Notes*. Up-to-date Release Notes and other documentation are posted on Oracle Technology Network (OTN) at this location:

  http://www.oracle.com/technetwork/documentation/agile-085940.html

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Configuring Content Synchronization and Syndication

This guide discusses basic configuration information for the Content Synchronization and Syndication application of Agile Product Lifecycle Management for Process. Topics in this manual include:

- Overview
- Content Synchronization and Syndication Settings
- Syndicating Custom Sections
- Best Practices and Use Cases
- XSD Locations
- Extensions
- National Language Support
- Common Tasks
- Related Guides

Overview

Using configuration files, you can limit or extend the behavior of the Content Synchronization and Syndication (CSS) application of your Oracle Agile Product Lifecycle Management for Process installation.

Agile configuration files are text-based XML files. Configuration settings can stand alone, or they can be organized within a nested set of XML elements.

The Agile configuration files are located at:

```
[X]: \%Prodika_Home%\Config
```

As of version 6.1, there are three subfolders and a few files under this location, as shown below.

```
[X]: \%Prodika_Home%\Config\Core
[X]: \%Prodika_Home%\Config\Custom
[X]: \%Prodika_Home%\Config\Extensions
[X]: \%Prodika_Home%\Config\environmentvariables.config
[X]: \%Prodika_Home%\Config\DeployedConfig.config
```

The `\%Prodika_Home%\Config` directories are defined as follows.

Table 1–1 Directory definitions

Directory	Description
Core	This directory holds files that should not be modified as part of the deployment. These system files have been encrypted as part of the installation to prevent accidental changes.
Custom	This directory holds the environment and feature configurations that can be modified.
Custom\Reference	This directory holds all of the core files that can be used as a reference when editing custom and extension files.
Extensions	This directory holds all of the files that can be modified to extend the behavior of the product suite. Refer to the *Oracle Agile Product Lifecycle Management for Process Extensibility Pack* for more information.

Commonly Edited Configuration Files

Several configuration files can be modified to affect application behavior or to model the application landscape, as listed in Table 1–2 below.

Table 1–2 Commonly edited configuration files, described

Configuration File Name	File Location	Description
environmentvariables.config	`%PRODIKA_HOME%\Config`	Defines variable settings for physical layout of servers and applications
EnvironmentSettings.config	`%PRODIKA_HOME%\Config\Custom`	Defines more advanced physical environment settings such as email addresses and service settings
CustomerSettings.config	`%PRODIKA_HOME%\Config\Custom`	Defines variables specific to the implementation requirements and provides the ability to override specific application feature configuration settings

CSS configuration is located in the following directory:

`Extensions/cssLibConfig.xml`

For information on other configuration files, refer to the *Agile Product Lifecycle Management for Process Configuration Guide*.

Content Synchronization and Syndication Settings

Overview

The `CssLibConfig.xml` configuration file contains settings needed to run the Agile Content Synchronization and Syndication (CSS) application. Most of the configuration is provided out-of-the-box, but some key settings vary.

CSS is an integration application that requires technical resources to configure for syndication.

There are two primary types of CSS syndication: internal and external.

Internal Syndication

The recommended architecture for doing internal syndication is for Agile to syndicate to a middleware product that can then be configured to send messages to one or many destination systems.

External Syndication

External syndication is used to publish specification data to a system outside of the customer firewall. The most common scenario is syndication to a third-party data exchange, such as 1Sync.

In an external syndication scenario, specific configurations to modify the message bundling, formatting, and transport are necessary.

Configuration Sections

ConfigSections

This section contains declarations of the configuration sections and configuration section handlers for the CSS configuration file. The ConfigSections node is core to the application configuration framework and should not be changed.

<NamedParams> Section

The **`<NamedParams>`** section is used to specify name-value pairs to be used elsewhere in the `CSSLibConfig` file or within the CSS application.

Key parameters:

Table 1–3 <NamedParams> section parameters, described

Parameter	Description
WorkflowFromEmailAddress	The "From" email used for workflow transition notifications
CssPortalActionItemsUrl	The URL used to build the link to Action Items in email notifications
ExcludeLinkTradeTypePkids	Specifies identifiers for trade specification types that should not be syndicated
SyndicationServiceUrl	This named parameter is used to configure the MessageTransporterFactory for a Web service enabled publication path. It represents the URL of the Web service endpoint that this CSS publication path will send messages to.

Table 1–3 <NamedParams> section parameters, described

Parameter	Description
APIResponseFolder	Used by the CSS service API in Extensibility Pack (from version 2.4.0.0.3) to specify the store location of the replied syndication messages from target system when it is using a delay response syndicate pattern such as AIA. Refer to the *Agile Product Lifecycle Management for Process Application Programming Interface Guide* for more information.

<CSSConfigurations> Section

The main portion of the configuration for CSS happens in this section. For each publication path, you specify how the messages will be created, organized, published, transported, and reconciled.

Each child node under **<CSSConfigurations>** represents a distinct CSS publication path. An individual publication path configuration will contain some or all of the following attributes.

Table 1–4 CSSConfigurations section attributes, described

Attribute	Description
CSS Config root node	Custom. Must correspond to the CSS publication namespace in the database configuration. The internal name of the publication path
Exchange	Custom. Can be used to specify the GLN of the target system when the publication path is for external syndication. The value is a unique identifier for a specific target system. Other than the requirement that the value be unique, there are no constraints for this value for internal data syndication. It is used with the transaction ID as a key to manage transactions and is sent along with the syndication message.
CssTransactionLoaderFactory	The transaction loader determines which TIPs should be published. The StandardCssTransactionLoaderFactory creates an object that loads all TIPs in the appropriate CSSPublicationNamespace that are in the configured workflow status: StagedForSyndication. (See "NamedWorkflowStatuses" on page 1-9.) This configuration should not typically be changed for the standard internal or external CSS syndication.
<MessageGenerators>	Contains a series of specific **<MessageGenerator>** nodes.

Table 1–4 CSSConfigurations section attributes, described

Attribute	Description
<MessageGenerator>	Contains a value attribute and a **<MessageGeneratorFactory>** and **<TipDataAdapterFactory>** node. This section maps specification types to a particular message generator configuration. There is one for each specification type that the publication path will support. This is a standard setting. The value attribute defines the four-digit specification type identifier that this **MessageGenerator** configuration is for. This configuration should not typically be changed for the standard internal or external CSS syndication.
<MessageGeneratorFactory>	Contains the path to the CSS **<MessageGeneratorFactory>** component and additional parameters, including the Export Model identifier (for example, CSSFormulationSpec) and the XSLTransformResolver. This configuration should not typically be changed for the standard internal or external CSS syndication.
<TipDataAdapterFactory>	The **<TipDataAdapterFactory>** component that is used to expose the data for syndication This configuration should not typically be changed for the standard internal or external CSS syndication.
<TipValidatorFactory>	This is a standard setting for using the default validation configuration.
<MessagePublisherFactory>	The publisher factory generates the XML for the transactions that were loaded by the configured CssTransactionLoader. Additionally, the configuration parameters to this factory allow the formatting and transformation of the message to be customized There are two message publisher factories to choose from. External syndications use the TransoraMessagePublisherFactory because it publishes every GLN (provider) separately but puts multiple transactions in each message. Internal syndications use the SingleTransactionPublisherFactory since it publishes each transaction in a separate message. This makes sense when you are working with an internal network because message granularity becomes more important than message quantity.

Table 1–4 CSSConfigurations section attributes, described

Attribute	Description
<MessageTransporterFactory>	The **<MessageTransporterFactory>** determines how the syndicated TIP message gets transported to the receiver. For internal syndication, the two most common options are the FileCopyMessageTransporterFactory and the SyndicationServiceTransporterFactory. The FileCopyMessageTransporterFactory saves the message to a specified location on a local or mapped drive as a file with a specified extension. The SyndicationServiceTransporterFactory sends the TIP message as a payload of a Web service request. This is a standard setting for the Web service based transport. Note that the object path has additional parameters, described on the next line.
MessageTransporterFactory object path parameters (for SyndicationServiceTransporterFactory)	[CSS name] | [Web service URL] (defined in the **<NamedParams>** configuration section) | [authentication method] | [target system user name] | [target system password] These parameters will need to be modified to reflect the target system user name and password.
<NamePaths>	Contains information about determining certain properties in the object model used for syndication. This configuration should not typically be changed for the standard internal or external CSS syndication.
<InfoProviderMap>	Contains one or more **<MapItem>** nodes, which represent a key-value pair of attributes. Example: `<MapItem key="GB" value="3155dc4be769-3a1e-4f89-98b4-2dd76124febb"/>` The key corresponds to a particular CSS target market. The value represents the Agile PKID (a unique identifier) for the specific information provider(s) that the publication path is related to.
<ResponseHandler>	Contains the configuration for how a syndication response message is handled. Responses may take the form of a web service response or a file drop. This configuration should not typically be changed for the standard internal or external CSS syndication.

Table 1–4 CSSConfigurations section attributes, described

Attribute	Description
<ResponseAdapterSettings>	There is currently only one ResponseAdapterFactory that can be configured for this setting: The CssResponseDataAdapterFactory. The ResponseAdapterFactory expects two parameters to be configured in its object URL string: ■ Publication Namespace: This is the name of the current publication path, as defined in the configuration. ■ ResponseDataAdapter: The ResponseDataAdapter is the class responsible for loading a response and transforming it into a usable object by CSS. Currently, there are three types of ResponseDataAdapters: ■ CssServiceResponseDataAdapter: Used for the standard Web service response message ■ CssResponseXMLDataAdapter: Used for some types of external syndication responses. ■ CssResponseDatabaseMDNDataAdapter: For 1-sync syndication, a specialized class that is able to read responses from a third-party database
	Specific Response Adapter Settings
ResponsesFolderLocation	The path to read responses from for reconciliation. This value should be changed to reflect the particular CSS environment.
ResponsesArchiveFolderLocation	The path to store inbound responses after processing. This value should be changed to reflect the particular CSS environment.
ResponsesEnvelopeXSLFile	Optionally, wrap the inbound response in another envelope by specifying an XSL transformation. This configuration should not typically be changed for the standard internal or external CSS syndication.
ResponsesFullTransformXSLFile	Optionally, transform the inbound response prior to processing. This configuration should not typically be changed for the standard internal or external CSS syndication.
<Responses>	This setting specifies which software component should process the incoming response. For the standard internal Web service enabled syndication, it is configured to retrieve and expose the Cross Reference values on the incoming response. This is a standard setting for Cross Reference-enabled integration.

Table 1–4 CSSConfigurations section attributes, described

Attribute	Description
<CssReconcilerFactory>	The **<CSSReconcilerFactory>** is the software unit that performs required actions based on the contents of the responses. Other than work-flowing the CSS TIP, the reconcilers do not have much else in common. The StandardTimedReconcilerFactory is used in situations in which the integration is asynchronous, so Agile cannot know when the responses will come in. This syndicate runs on a certain schedule and reconciles received responses and takes action to work-flow the TIPs that have not received responses in a timely manner. The StandardTimedReconcilerFactory works in conjunction with the StandardCssResponseFactory that is configured in the **<Responses>** XML node. The StandardCssReconcilerFactory gets the results from the configured Adapter and workflows the TIP accordingly (Active/Failed). The CrossReferenceReconcilerFactory is called both directly from the syndicate that is publishing the TIP and also from a separate syndicate. It can therefore be used in synchronous and asynchronous transactions. Additionally, it looks inside the responses for an ID that the specification can also be known by: a cross-reference. This cross-reference is usually an ID internal to and assigned by the target system. The CrossReferenceReconcilerFactory works in conjunction with the CrossReferenceResponseFactory, which is configured in the **<Responses>** node. The RepeatableCrossReferenceReconcilerFactory (introduced from ver6.0.0.3.48) can retry the syndication if one attempt was failed. A parameter NameValuePair:NumberOfRetries can control the number of retry times. For the successful or pending syndication, it works with the same behavior as CrossReferenceReconcilerFactory. For the standard internal Web service enabled syndication, the configured component will set the cross-reference on the source specification. This is a standard setting for cross-reference-enabled internal integration.
<ValidatorDelegates/>	The ValidatorDelegates setting is used to aid the XML validation process by calling out to custom classes to perform validation that cannot be accomplished via the normal process. This setting is not used for the standard internal or external CSS syndication.
<ConversionDictionary/>	The ConversionDictionary setting is used to convert specified TIP properties to a desired unit of measurement. This setting is not used for the standard internal or external CSS syndication.

PublicationPathConfigs

Configuration of the transformations and validation occurs here.

The reason that the PublicationPathConfigs are in a separate location from the rest of the configurations is that they are target market specific. You can have a different PublicationPathConfig for each publication path in a specific target market. These are listed in the **<CssConfigurations>** section.

Each PublicationPathConfig contains the following sections:

Exchange

The exchange GLN needs to match the GLN configured in the **<CssConfigurations>** section for that publication path.

TargetMarket

This is typically the two-character country ID of the target market for this publication path.

TradingPartner

Usually this is configured as "default" for both the name and GLN. If there are multiple pub path configurations (that is, different messages for different target markets), then each section should be configured independently.

XSLTransformMaps

Used to indicate which XSL transformation should be applied based on the transaction command type.

Validator

If the TIP should be validated before it can be work-flowed or published, this is where the validator would be configured.

If the publication path is going to be used for more than one specification type, you can use the MappedTipValidatorFactory to configure different validators for each specification type.

NamedWorkflowStatuses

This section contains a list of CSSWorkflowStatus PKIDs and names. The names are then referred to instead of the PKIDs. Additionally, the Transora Publisher uses the names to determine what step to workflow the TIP to based on the response received after syndication.

The values in this section should not be changed unless the core CSS workflow status identifiers change.

Syndicating Custom Sections

The `exportExtensions.xml` file contains settings needed to add custom section data to the specification syndication. This file must be modified to specify the custom sections that should be syndicated for each specification type.

Configuration Sections

Config Sections

This section contains declarations of the configuration sections and configuration section handlers for the CSS configuration file. The ConfigSections node is core to the application configuration framework and should not be changed.

<ExportExtensibilityHandlers> Section

This section contains the configuration options for syndicating custom section data.

<SpecType#> Section

A **`<SpecType#>`** node is required for each specification type that is to be syndicated. For instance, to syndicate custom section data for an material specification, a **`<SpecType1004>`** node must exist.

Each **`<Spec Type#>`** section contains ExtensionItem nodes that determine how the custom section data output should appear in the syndication. Two different handlers are available:

1. **XmlNodeCreationExtensibilityHandler** — Creates a simple XML node used to wrap any other data. The name of the XML element created is indicated after the `$` symbol. Appending a pipe (`|`) symbol and the term "CloseTag" will create the closing XML tag.
2. **CustomSectionExtensibilityHandler** — Used to produce the custom sections data. Only the custom section provided using a pipe-delimited list of custom section numbers will be syndicated.

Example configuration

```
<SpecType1004>

  <ExtensionItem
handler="Class:Xeno.Prodika.ExportExtension.XmlNodeCreationExten
sibilityHandler,

ExportExtension$ProdikaExtensibility" />

<ExtensionItem
handler="Class:Xeno.Prodika.ExportExtension.XmlNodeCreationExten
sibilityHandler,

ExportExtension$ProdikaExtensibilityItem" />

<ExtensionItem
handler="Class:Xeno.Prodika.ExportExtension.XmlNodeCreationExten
sibilityHandler,

ExportExtension$ProdikaCustomSections" />

<ExtensionItem
handler="Class:CustomSectionXMLLib.handlers.CustomSectionExtensi
bilityHandler,
```

```
CustomSectionXMLLib$1000123|1000456|1000789" />

<ExtensionItem
handler="Class:Xeno.Prodika.ExportExtension.XmlNodeCreationExten
sibilityHandler,ExportExtension$ProdikaCustomSections|CloseTag"
/>

<ExtensionItem
handler="Class:Xeno.Prodika.ExportExtension.XmlNodeCreationExten
sibilityHandler,ExportExtension$ProdikaExtensibilityItem|CloseTa
g" />

<ExtensionItem
handler="Class:Xeno.Prodika.ExportExtension.XmlNodeCreationExten
sibilityHandler,ExportExtension$ProdikaExtensibility|CloseTag"
/>

</SpecType1004>
```

...will result in the following XML being added to syndication:

```
<ProdikaExtensibility>
   <ProdikaExtensibilityItem>
      <ProdikaCustomSections>
         <CustomSection1000123>...custom section data...
</CustomSection1000123>
         <CustomSection1000456>...custom section data...
</CustomSection1000456>
         <CustomSection1000789>...custom section data...
</CustomSection1000789>
      <ProdikaCustomSections>
   <ProdikaExtensibilityItem>
<ProdikaExtensibility>
```

...where **`<CustomSection1000123>`** is the name of the custom section, and `...custom section data...` is the content of the custom section.

Best Practices and Use Cases

How to Configure Data Syndication to a Web Service or a File

The **<MessageTransporterFactory>** setting determines how the syndicated TIP message gets transported to the receiver. To configure data syndication to a web service or a file, set the transporter to either SOAP or File Copy as follows:

Example configuration

For SOAP transport:

```
<MessageTransporterFactory
value="Class:Xeno.Prodika.CSS.MessageTransport.Soap.Http.Syndica
tionServiceTransporterFactory,CSSLib$WSSyndication|SyndicationSe
rviceUrl|<value>|prodika|prodika"/>
```

where <value> is one of the following:

- NoAuthenticationEndpoint—Plain text message with No authentication transport
- HttpBasicEndpoint—Plain text message with HTTP Basic authentication transport
- WSSUsernameTokenEndpoint—Encrypted message with WSS 1.1 username token authentication transport

For File transport:

```
<MessageTransporterFactory
value="Class:Xeno.Prodika.CSS.MessageTransport.FileCopyMessageTr
ansporterFactory,CSSLib$c:\data\css\qa\out|.xml"/>
```

How to Configure the Authentication Methods

Agile PLM for Process supports three types of authentication methods to protect the communication between CSS and the integration endpoint of the receiver application.

The supported authentication methods are:

- No Authentication
- HTTP Basic Authentication
- WSS1.1 Username Token Authentication

Configuring the No Authentication Method

This method doe not protect the session between the client and server in the syndication process. The message is sent by plain text.

1. Open the configuration file: <Prodika_Home>\Config\Extensions\cssLibConfig.xml and locate the following line:

   ```
    <MessageTransporterFactory
   value="Class:Xeno.Prodika.CSS.MessageTransport.Soap.Http.Synd
   icationServiceTransporterFactory,CSSLib$WSSyndication|Syndica
   tionServiceUrl|<endpoint_name>|| "/>
   ```

2. Make sure this line is not commented and is the only uncommented tag with name "MessageTransporterFactory" node.

3. Replace the <endpoint_name> with **NoAuthenticationEndpoint**.
4. Save this configuration file and restart the Remoting Container service.

Configuring the HTTP Basic Authentication Method

This method enables the HTTP Basic Authentication when CSS sends the syndication request to the service. The message is sent by plain text.

1. Open the configuration file: <Prodika_ Home>\Config\Extensions\cssLibConfig.xml and locate the following line:

```
 <MessageTransporterFactory
value="Class:Xeno.Prodika.CSS.MessageTransport.Soap.Http.Synd
icationServiceTransporterFactory,CSSLib$WSSyndication|Syndica
tionServiceUrl|<endpoint_name>|<username>|<password> "/>
```

2. Make sure this line is not commented and is the only uncommented tag with name "MessageTransporterFactory" node.
3. Replace the <endpoint_name> by **HttpBasicEndpoint** and fill in the username and password which are used to access the protected web service into the <username> and <password> fields.
4. Save this configuration file and restart the Remoting Container service.

Configure the "WSS 1.1 Username Token Authentication" Method

This method enables the WSS (1.1) Username Token Authentication when CSS sends the syndication request to the service. The message will be encrypted by the certificate issued by the service. To accomplish this method, you must first get the certificate issued by the syndication target service provider.

1. Import the certificate issued by the service provider into a applicable path of the machine hosting CSS application.
2. Open the configure file: <PLM4P_Home>\Config\Extensions\cssLibConfig.xml and locate the following line:

```
<MessageTransporterFactory
value="Class:Xeno.Prodika.CSS.MessageTransport.Soap.Http.Synd
icationServiceTransporterFactory,CSSLib$WSSyndication|Syndica
tionServiceUrl|<endpoint_name>|<username>|<password> "/>
```

3. Make sure this line is not commented and is the only uncommented tag with name **"MessageTransporterFactory"** node.
4. Replace the <endpoint_name> by **WSSUsernameTokenEndpoint** and fill in the username and password which are used to access the protected web service into <username> and <password> fields.
5. Save this configuration file and then open:

```
PLM4P_Home >\RemotingContainer\RemotingContainer.exe.config
```

6. Locate the following section:

```
<behaviors>
      <endpointBehaviors>
        <behavior name="WSSecureBehaviour">
          <clientCredentials>
            <serviceCertificate>
              <defaultCertificate
                  findValue="<value>"
                  storeLocation="<location>"
                  storeName="<name>"
                  x509FindType="<type>"/>
            </serviceCertificate>
          </clientCredentials>
        </behavior>
      </endpointBehaviors>
    </behaviors>
```

7. Fill the values of the fields in the above section according to the certificate information and the store path in step 1. For the more information about the configuration, please refer to Microsoft document: http://msdn.microsoft.com/en-us/library/aa347741.aspx

8. Locate the following section in the same file:

```
<endpoint
   name="WSSUsernameTokenEndpoint"
   address=""
   binding="customBinding"
   bindingConfiguration="WSSUsernameTokenBinding"
   contract="SyndicationServiceInterface"
   behaviorConfiguration="WSSecureBehaviour">
        <identity>
             <dns value="<service_host_name>"/>
        </identity>
</endpoint>
```

9. Enter the host name of the server which is hosting the syndication target service into the <service_host_name> field.

10. Save the configuration and then restart the Remoting Container service.

XSD Locations

The CSS schema files are located in:

`PRODIKA_HOME\schema\syndication`

The root schema file is:

`css.xsd`

Extensions

For a discussion on how to extend CSS, refer to the Agile PLM for Process extensibility documentation.

National Language Support

For a discussion on CSS support of National Language Support (NLS), refer to the *Agile Product Lifecycle Management for Process Install/Upgrade Guide*.

Common Tasks

The following common tasks are explained below:

- Changing a message by creating a custom transformation
- Changing a message by writing a transporter
- Customizing response handlers

Changing a Message by Creating a Custom Transformation

You can create a custom message by creating an eXtensible Stylesheet Language Transformation (XSLT). By creating a custom XSLT, you can customize the outbound message to conform to the target system.

Changing a Message by Writing a Transporter

A transporter delivers a message to the end point. A custom transporter can be created by implementing the `IMessageTransporterFactory` and `IMessageTransporter Interfaces`. To use a custom transporter, change the `MessageTransporterFactory` setting in the `CSSLib.config` file.

Customizing Response Handlers

Response handlers take a response from an external system and translate it into something usable by Oracle Agile PLM for Process. By creating a custom response handler, you can customize the behavior of the application depending on the response from the target system.

For example, you may want to update cross reference information and workflow syndicated specification to a specified status based on the response message from the target system. Or, you could log an event to perform audit information.

In order to create a custom response, you must implement the following interfaces.

- ICssReconciler
  - Manages the reconciliation process

  - Moves Tips to the appropriate workflow status based on responses
- ICssResponseDataAdapter
  - Creates a response XML, consumed by `ReponseHandler`
- ICssResponseHandler
  - Creates a collection of `ICssReponse`, consumed by the reconciler
- ICssResponse
  - Contains response information for the reconciler

Related Guides

Refer to the following guides for more information on CSS and related tasks:

- *Agile Product Lifecycle Management for Process Content Synchronization and Syndication User Guide*—General information about the CSS application
- *Agile Product Lifecycle Management for Process Configuration Guide*—Configurations for Agile PLM for Process
- *Agile Product Lifecycle Management for Process Security Configuration Guide*—Security-related configurations for Agile PLM for Process
- *Agile Product Lifecycle Management for Process Install/Upgrade Guide*—Installing National Language Support
- *Agile Product Lifecycle Management for Process Data Administration Toolkit Guide*—Widgets available for administrative tasks
- *Agile Product Lifecycle Management for Process Application Programming Interface User Guide*—Web services available through the API
- *Agile Product Lifecycle Management for Process Administrator User Guide*—CSS workflows