Oracle® Agile
Engineering Data Management

e6.1.2 Web Services Manual

Part No. E24278-01

August 2011

Copyright and Trademarks

CONTENTS

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 e6 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 e6 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 e6 can be found in the Release Notes file on the Oracle Technology Network (OTN) Web site (http://www.oracle.com/technology/documentation/agile_eseries.html)

Oracle 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 to Agile e6 Web Services

About Web Services

Web Services are technologies for building distributed applications. These services, which can be made available over the Internet, use a standardized XML messaging system and are not tied to specific operating systems or programming languages. Through Web Services, companies can encapsulate existing business processes, publish them as services, search for and subscribe to other services, and exchange information throughout and beyond the enterprise. Web Services are based on universally agreed upon specifications for structured data exchange, messaging, discovery of services, interface description, and business process design.

A Web Service makes remote procedure calls across the Internet using:

- HTTP/HTTPS or other protocols to transport requests and responses.
- Simple Object Access Protocol (SOAP) to communicate request and response information.

The key benefits provided by Web Services are:

- **Service-oriented Architecture** – Unlike packaged products, Web Services can be delivered as streams of services that allow access from any platform. Components can be isolated; only the business-level services need be exposed.
- **Interoperability** – Web Services ensure complete interoperability between systems.
- **Integration** – Web Services facilitate flexible integration solutions, particularly if you are connecting applications on different platforms or written in different languages.
- **Modularity** – Web Services offer a modular approach to programming. Each business function in an application can be exposed as a separate Web Service. Smaller modules reduce errors and result in more reusable components.
- **Accessibility** – Business services can be completely decentralized. They can be distributed over the Internet and accessed by a wide variety of communications devices.
- **Efficiency** – Web Services constructed from applications meant for internal use can be used for externally without changing code. Incremental development using Web Services is relatively simple because Web Services are declared and implemented in a human readable format.

Core Technologies

Oracle's Agile e6 Web Services use industry standard core technologies. These are:

1. Web Services Description Language (WSDL)
2. XML and XML Schema
3. Simple Object Access Protocol (SOAP)

Web Services Description Language (WSDL)

WSDL is an XML-based format for describing the interface of a Web Service. WSDL describes the endpoints, location, protocol binding, operations, parameters, and data types of all aspects of a

Web Service:

- The WSDL that describes a Web Service has the following characteristics:
  - It is published by the service provider.
  - It is used by the client to format requests and interpret responses.
  - It can be optionally submitted to a registry or service broker to advertise a service.
- Additionally, WSDL describes the following:
  - The operations that are provided by a Web Service.
  - The input and output message structures for each Web Service operation.
  - The mechanism to contact the Web Service.

XML and XML Schema

A WSDL file is published an XML file. Document/Literal is required as part of the WS-I interoperability standard. This standard sets the basis for modern Web Service usage.

- **Document** – The payload for an operation, however complex, must be defined in a single XML element.
- **Literal** – The definition of single XML element must be described by an XML Schema embedded in the WSDL file.

When using Document/Literal formatting, the WSDL file will contain an XML Schema definition that defines all messages and data types that are used for a particular service. The payload itself will consist entirely of XML data structures.

Simple Object Access Protocol (SOAP)

SOAP is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. SOAP uses XML to define an extensible messaging framework.

SOAP messages consist of the following:

- An envelope for wrapping messages, including addressing and security information.
- A set of serialized rules for encoding data types in XML.
- Conventions for a procedure call and, or response.

Web Services Architecture

You can view Web Services architecture in terms of roles and the protocol stack:

- **Roles**:
  - **Service provider** – This provides the service by implementing it and making it available on the Internet.
  - **Service requester** – This is the user of the service who accesses the service by opening a network connection and sending an XML request.
  - **Service registry** –This is a centralized directory of services where developers can publish new services or find existing ones.
- **Protocol Stack**:
  - **Service transport layer** – This layer uses the HTTP protocol to transport messages between applications.

- **XML messaging layer** – This layer encodes messages in XML format using SOAP to exchange information between computers. It defines an envelope specification for encapsulated data that is transferred, the data encoding rules, and remote procedure call (RPC) conventions.
- **Service description layer** – This layer describes the public interface to a specific Web Service using the Web Service Description Language (WSDL) protocol. With WSDL, it defines an XML grammar to describe network services. The operations and messages are described abstractly, and then bound to a network protocol and message format. WSDL allows description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.
- **Service discovery layer** – This layer centralizes services into a common registry using the Universal Description, Discovery, and Integration (UDDI) protocol. UDDI is a platform-independent, XML-based registry for businesses worldwide to list themselves on the Internet.

About Agile e6 Web Services

Agile e6 Web Services expose a subset of the Product Lifecycle Management (PLM) functionalities of the Agile e6 Application. These services support functionalities provided by PLM modules in Agile e6 application, such as Item Management, Project Management and many other functions of Agile e6.

Implementation of Agile e6 Web Services adheres to the following principles:

- Well defined, standards based discoverable Interface
- Java based Web Services Framework using Oracle WebLogic
- Modularized Agile e6 Schema (XSD) and WSDL for easy maintenance
- Standards-based WSDL to ensure compatibility across various clients (.NET, Java, and BPEL)
- Bulk APIs wherever applicable for better performance

The Core Web Services

Agile e6 Core Web Services is a set of services for the following functionalities:

1. Business Object Web Services
2. Document Management Web Services
3. MetaData Web Services
4. Configuration Web Services

About Agile e6 Web Services Framework

The Web Service Framework is an additional layer on top of Agile e6, which supports inbound and outbound communication based on standard Web Services technology. It provides the means to call External Web Services from inside Agile e6 LogiView Procedures (outbound direction). In addition, it allows external applications (Web Service Clients) to call the Agile e6 APIs through Web Services.

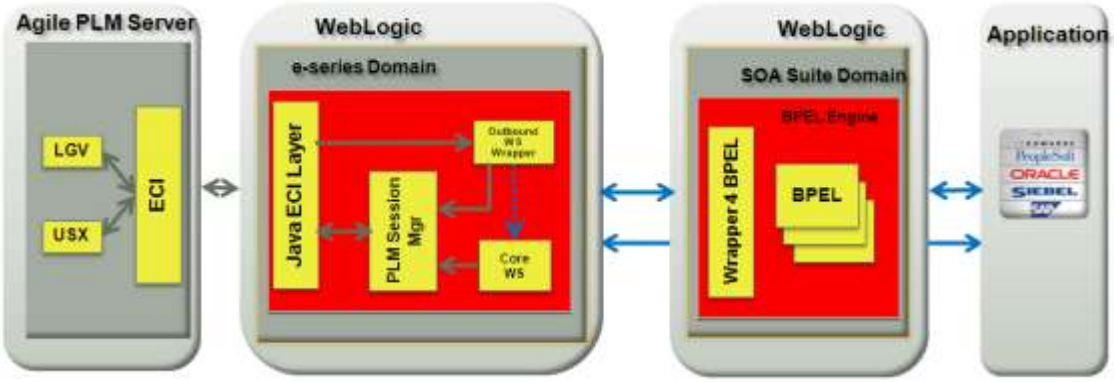
The Web Service Framework comes with a set of predefined *core* web services, which, out of the box, support the most common integration scenarios like *create PLM object* or *get PLM object*.

In a future release, the web services framework will be enhanced with a customizing framework. This will allow the implementation of custom web services, which funnel inbound web service calls into API call to the Agile e6.

Components of Agile e6 Web Services Framework

The e6 Web Services Framework comprises of the following:

- **Web Service Wrapper** – to support outbound web service calls from LogiView Procedures.
- **Core Web Services** – to support inbound web service calls mapped into ECI-API calls.



Chapter 2

Getting Started with Agile e6 Web Services

Prerequisites

Agile e6 Web Services are deployed on an Agile e6 WebLogic application domain.

To use the Agile e6 Web Services Framework for inbound and outbound Web Services based business data transaction, you are required to ensure the following:

- Operational Environment is set:
  - Oracle Agile e6, Release 6.1.2 plus HF1 or higher is installed.
  - WebLogic Server is installed and
- Web Services Framework is configured for the following:
  - Authentication Provider in WebLogic.
  - Web Service Security.
- Test the Inbound Web Services.

Then you will be able to call the available Core Services or implement your own outbound Web Service Wrappers.

Operating Environment

Agile e6 Application	Release e6.1.2 plus HF1 or higher
Default Web Services Engine	Oracle WebLogic Server Note: The version of this server is the one that is released with the Agile e6 application.
Java 2 Platform Standard Edition Development Kit	6.0

Web Services Engines

All Application Server vendors, such as Oracle, IBM, have built-in Web Services infrastructure solutions that are integrated with their application servers. For non-web services integrated applications, there are stand-alone products, such as AXIS from Apache, which provide Web Services infrastructure that can be integrated with different application servers.

The Agile e6 Web Services Framework works on the following Web Service Engines:

- Oracle Apps Server Web Service Infrastructure
- Oracle SOA Suite
- WebLogic Web Service Infrastructure
- Axis version 2.0 to support JAX-WS features, especially the MTOM

Web Service Development Tools

To develop your own Web Services, you can use various tools. Development platforms vary in their SOAP implementations. Implementation differences in certain development platforms may prevent access to some or all of the features in the API. If you are using Visual Studio for .NET development, it is recommended that you use Visual Studio 2003 or higher.

In this documentation the development and test of Web Services is described using Oracle's **JDeveloper.** You can also use Oracle's Enterprise Pack for Eclipse (EPE).

For complete information on JDeveloper and Enterprise Pack for Eclipse, refer to Oracle Technology Network web site: http://www.oracle.com/technetwork/developer-tools/index.html

The Agile e6 web security framework provides tool objects (source codes) to retrieve security information from the web service call which can be used to initiate a PLM session.

Standards Compliance

The Agile e6 Web Services are implemented in compliance with the following standards:

Standard	Location
Simple Object Access Protocol (SOAP) 1.1/1.2	http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
Web Service Description Language (WSDL) 1.2	http://www.w3.org/TR/2001/NOTE-wsdl-20010315
WS-I Basic Profile 1.1	http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html
XML Schema 1.1	http://www.w3.org/XML/Schema
SOAP Message Transmission Optimization Mechanism (MTOM)	http://www.w3.org/TR/soap12-mtom/
JAX-WS 2.0/2.1/2.2 (JSR 224)	http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/
JAXB 2.0/2.1/2.2 (JSR-222)	http://jaxb.java.net/

Understanding the Agile e6 Web Services Authentication and Performance

In the implementations where scalability is critical, a lightweight context management facility for authentication is available and its use is recommended. With this facility, authentication is managed using a combination of *user credentials* and a `sessionID` token (the standard HTTP session ID maintained by the web container):

- When user credentials are presented in the SOAP header of a Web Service request, formal authentication is performed prior to the application execution of the Web Service operation. If the authentication succeeds, the operation proceeds and a special SessionID token is placed in the SOAP header of the Web Service reply.
- Whenever the `sessionID` is included by the client in subsequent Web Service requests, that `sessionID` will be used to restore cached session information, thus bypassing the substantially more time consuming process of re-executing the authentication.

  Note that when presented with both the `sessionID` and a valid set of user credentials, an

attempt will be made to use the `sessionID` before resorting to the user credentials and re-authentication. As expected, the session that is being tracked by the `sessionID` is subject to expiration and other security checks.

The facility is a distinct alternative to the basic authentication standard described by Web Services Security. Using the `UserName` token as provided in WS-Security, while fully supported as part of Agile e6 WSI Basic Profile compliance, will not yield the same benefit as using the higher-performance session optimization facility provided by the Agile e6 implementation.

The Agile e6 PLM Session Handling

Every call of an Agile e6 Core Web Service needs an Agile e6 PLM server instance. It is very important to limit the number of e6 server instances to reduce the resource loading on the server. This is handled by the following mechanisms:

- HTTP Session – The web service first tries to find an e6 server instance assigned to the HTTP session of the current web service call. If it is found, the web service call uses the existing e6 server instance.
- PLM Ticket – A PLM ticket is returned in the response of a core web service operation. This ticket can be used to access the same e6 server instance that created the ticket.

  While authenticating a web service call, if a ticket is passed instead of the password, the session manager uses the e6 server instance, even if no e6 server is assigned to the HTTP session.

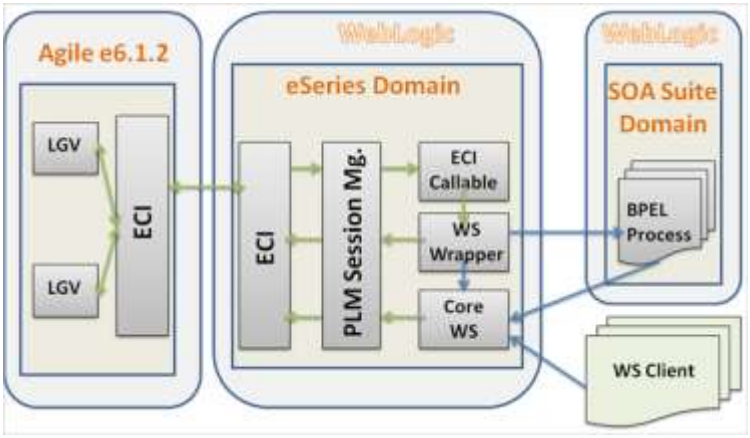
  The PLM Ticket mechanism is the only way to let the calls to different core web services, such as Metadata or BusinessObject web services use the same e6 server instance.

  A web service client should use the PLM ticket as soon as it has called the first operation.

  This is the only way to share an e6 session between two different Core Services.

To free an Agile e6 server instance assigned to a web service session, the client calls one of the *closeSession* operations (every core web service provides this function) with the PLM ticket as the *password*. This shuts down the Agile e6 server instance and frees up the server resources.

The Agile e6 PLM Session Manager



The Agile e6 PLM Session Manager lets you manage the PLM Session objects which are used to keep the existing connections and user contexts to the Agile e6 server.

The key to an existing PLM Session object is the **session ID**, which is generated by PLM Session Manager.

The **PLM Session** provides connection to Agile e6 server.

To retrieve a PLM Session, *PlmTicket* is provided. When a new PLM Session is created, the PlmTicket is set to the PLM Session, which is then set into the SOAP message to the client side.

The life cycle of PLM Session is as same as the given *HttpSession*. The timeout of an *HttpSession* is specified in **web.xml** with the following information:

```
<session-config>
            <session-timeout><time-in-second></session-timeout>
</session-config>
```

The PLM Ticket

A Response contains a string that can be used in subsequent calls. This string is called the *PLM Ticket*. The ticket gives the caller access to the same Agile e6 instance (PLM server) that was used in the last request. The ticket is remains valid only as long as the Agile e6 instance is running. After obtaining a ticket, the client code needs to configure the port by setting the ticket string as password. See *BindingProvider* in the example given under Authenticating in a Web Service Client.

The PLM Ticket improves the web services performance and simplifies the session management, If different Web Services are used in a use-case flow, which is very likely, the ticket returned by the response(s) of one service operation (say *Configuration.setUserContext)* is used as a password when client makes a call for another service operation (say *BusinessObject.getObjects)*.

The ticket sharing among different client ports eliminates the need for the server to start new Agile e6 sessions, which would result in new Agile e6 servers being started.

Understanding the Agile e6 Web Services Requests

In the Agile e6 web services framework, each operation has its own request data type, which is inherited from *RequestHeaderType*. The *RequestHeaderType* for all the requests has only the following elements:

- **messageId** (String, optional): Default value for the ID is the current system time in milliseconds.
- **messageName** (String, optional): Default value for the message name is the simple class name.

Obtaining the Agile e6 Metadata

You can obtain the basic Agile e6 metadata through Agile e6 Java client. Look for the data model of the Agile e6 application.

To obtain Agile e6 metadata through a web services operation, use the Metadata service. This service requires an entity name and a mask name.

See Configuration Parameters for further information.

Understanding the Agile e6 Web Services Responses

The ResponseHeaderType has the following members:

messageId (String, required)	Default value for the ID is the current system time in milliseconds
messageName (String, required)	Default value for the message name is the simple class name
statusCode (ResponseStatusCode, required)	Default value for the status code is SUCCESS
exceptions (List<PlmExceptionType>, optional): warnings (List<PlmWarningType>, optional)	
ticket (String, optional)	

Response Status Code

The response obtained from every Web Service call contains a response **statusCode**, which indicates the success or failure of a Web Service operation.

These Response Status Codes are of four types:

SUCCESS	Indicates that all Web Services in the batch were executed successfully and that all operations worked as intended.
FAILURE	Indicates that all Web Services in the batch failed during execution, indicating the intended operations were not performed.
WARNING	Indicates that while Web Services in the batch were successfully executed, however certain warnings were also encountered during the execution. These warnings need to be analyzed by the client to verify that all operations worked as intended.
PARTIAL_SUCCESS	Indicates a partial success in the execution of batch Web Services when one or more but not all batch requests have failed. Even if a single Web Service fails among a batch of Web Services, the response status code will indicated PARTIAL_SUCCESS.

White-list Mechanism for Masks

To ensure that only the masks designed for the access of web services are used, all the mask names are checked against a White-list that is maintained by the administrator of the Agile e6 installation.

List of Mask Names

A configuration rubric named EDB-WSI-MASKS is used. It contains sub-parameters such as EDB-ARTICLE-WSI and each of these sub-parameters contain a mask name rule pattern. All rules for an entity are checked for an operation that involves an item as an entity (or as parent entity in case of a relation).

Important It is recommended to implement special masks designed for web services, instead of only adding the standard masks like EDB-ART-SLI (or "*") to the white list.

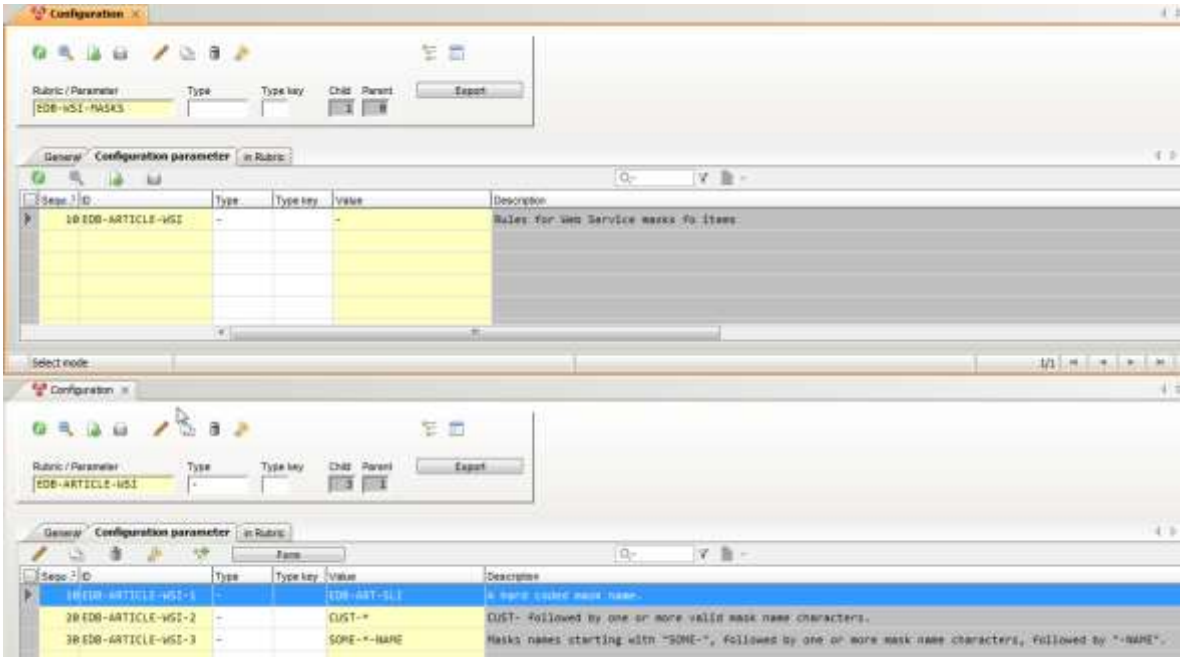
Adding standard masks can have certain implications due to the following:

- The security check for masks is not strong enough.

- To be able to access invisible fields, you are required to first make them visible. These should be done as these are not opened to the customer.
- Performance suffers as the standard masks contain too many fields.

Configuration Parameters

The Configuration Parameters are entered as shown in the image below.



The rule pattern consists of valid mask name characters and may contain one or more asterisks (*) to indicate one or more mask name characters.

The web service session reads these configuration rules and checks each combination of entity and mask name passed by a client against the rules for the respective entity. If one of the rules accepts the mask name, access is granted, If not, the access to the mask is denied and an *IllegalAccessException* is thrown and marshaled back to the client.

The rules are cached by the session so that the subsequent operations don't have the overhead of reading the rules again. The rules can also be cached in the Business Service, for instance by enhancing the Permission Manager. This provides a domain wide cache, instead of a session based cache.

Exceptions and Warnings

The Agile e6 framework throws an exception (WebFault) only if a severe technical problem occurs, for instance, in an event of a connection loss to the Agile e6 server. When an operation is not successful, the system will throw an Exception or a Warning.

- In case of **FAILURE**, an exception is issued, while a warning may or may not be issued.
- In case of **WARNING**, only a warning is issued.

When the status is WARNING, the outcome of the operation is unknown. You are required to check it manually whether the operation was successful or not.

Counting the Objects

A count request is indicated by the flag **countOnly** in the query request object. The service then executes a count (which ignores the mask limit) and returns a pseudo **PlmObject** with a **COUNT** attribute and a **RECORD_LIMIT** attribute.

The value of the **COUNT** attribute is an Integer with the number of objects matching the query, and the value of the **RECORD_LIMIT** attribute is the record limit of the mask used for the count operation.

Chapter 3

Setting up the Agile e6 Web Services Infrastructure

Installing the Agile e6 Web Services Framework

The Agile e6 Web Services Framework is installed during the basic installation of Agile e6 application. The WebLogic domains are created with the Agile e6 application installer and/or the Agile e6 Administration Client. By default, the installation has two WebLogic domains – one for an installation and an additional one for every application. Web Services are deployed on the application specific WebLogic domain.

For complete details, please refer to the *Agile e6 Installation and Administration Documents*, besides the *Agile e6 Hot-fix Readme*.

Creating the WebLogic Agile e6 Domain

You are required to obtain a WebLogic Agile e6 Domain name from the Agile e6 Administrator, or create a new domain. This is required as the Web Services are deployed into this WebLogic e6 domain.

During the installation of the WebLogic Server two domains are created. Each domain consists of an **AdminServer** and an **eSeries-01** server. The AdminServer is only for the administration of the domain, while the eSeries-01 contains the Agile e6 deployments.

For example, the domains directory *C:\Oracle\Middleware\user_projects\domains* contains the domain names *eSeries_domain* and *eSeries_domain_plmref*.

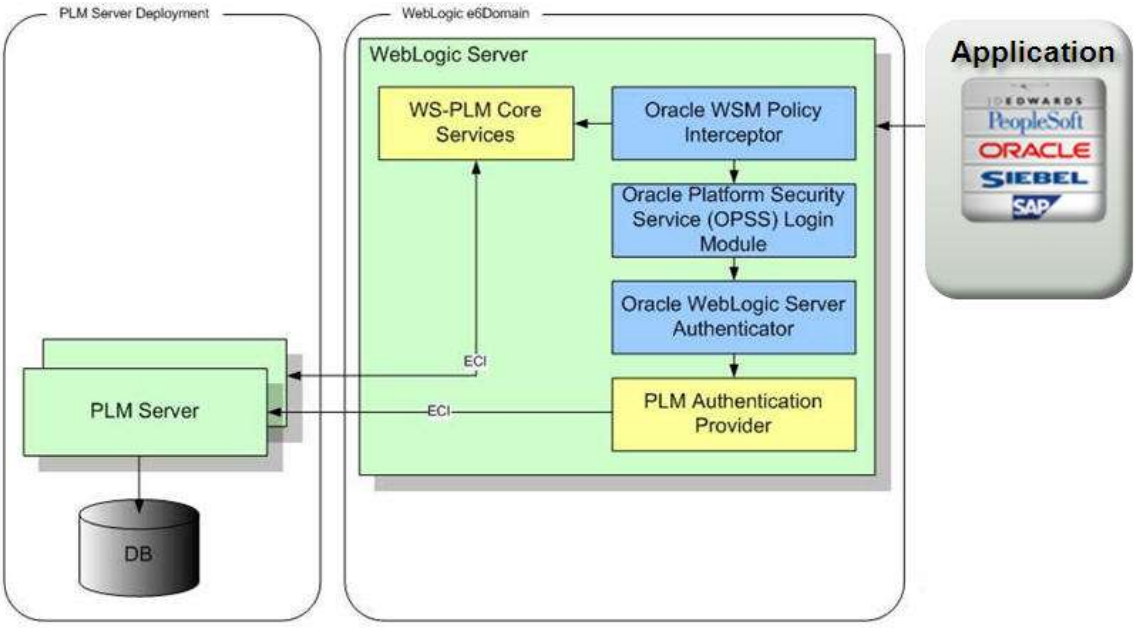
For complete details on how to install the WebLogic server and create a domain, please refer to the installation manual *Agile e6.1.2 Server on Windows/Unix*.

Configuring the PLM Authentication Provider in the WebLogic Server

The Agile e6 *PLM Authentication Provider* is required to authenticate the incoming Web Requests. It is also used for providing the *PLM user name* and *PLM Password* to the web services to connect to the Agile e6 application.

The authentication provider is called by the *WebLogic Security Framework*. The *Security Realm* of the domain has to be configured to use the Agile e6 PLM Authentication Provider.

The following diagram depicts the authentication mechanism:



The PLM authentication provider uses the standard mechanism of the WebLogic server and can be configured with the WebLogic Administration Console.

The Authentication Provider is installed at the time of Agile e6 application installation. During this installation, the PLM User and PLM password are set to an empty value. These cannot be set with the batch installer, hence, they have to be added manually with the WebLogic Console. For complete details on Agile e6 application installation, including the preliminary settings of the authentication provider, refer to the *Agile e6 Server Installation Guides*.

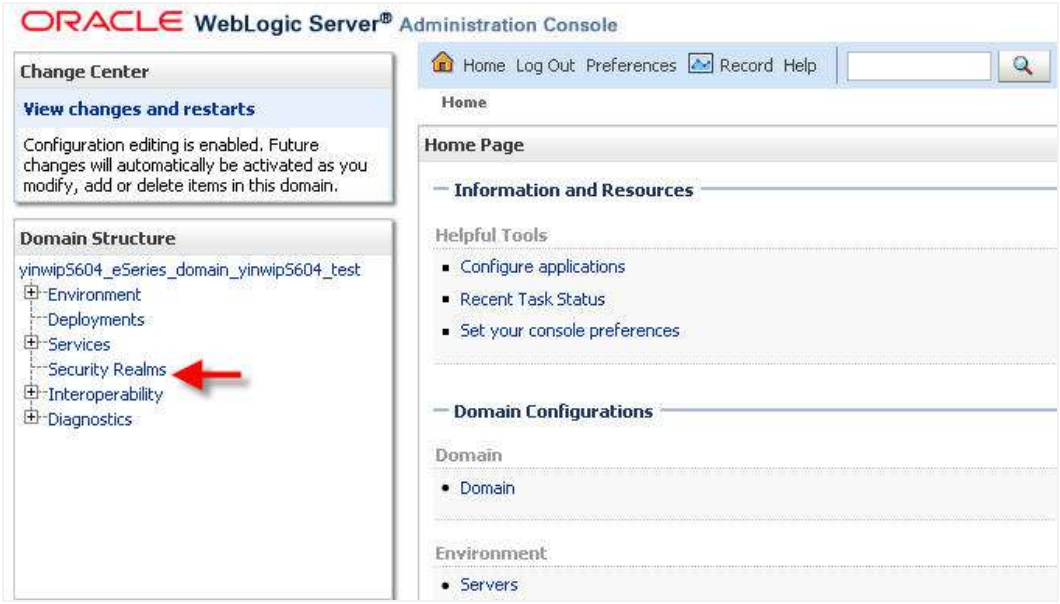
Note	The user should be a special user with no manager rights.

To configure the Agile e6 Authentication Provider:

Note	The application domain has to be configured while the installation domain remains unchanged.

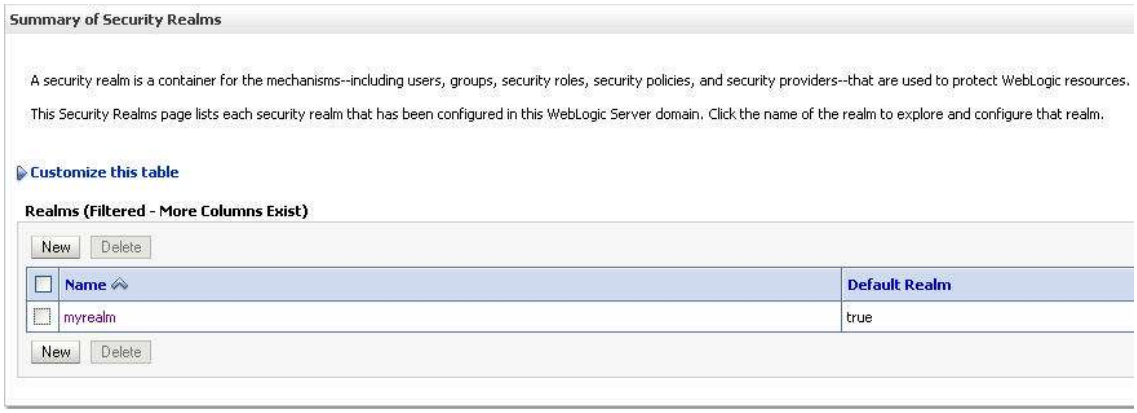
1. Log in to the Administration Console for the application domain of the Oracle WebLogic Server.

   Example: `http://server:7405/console/login/LoginForm.jsp`



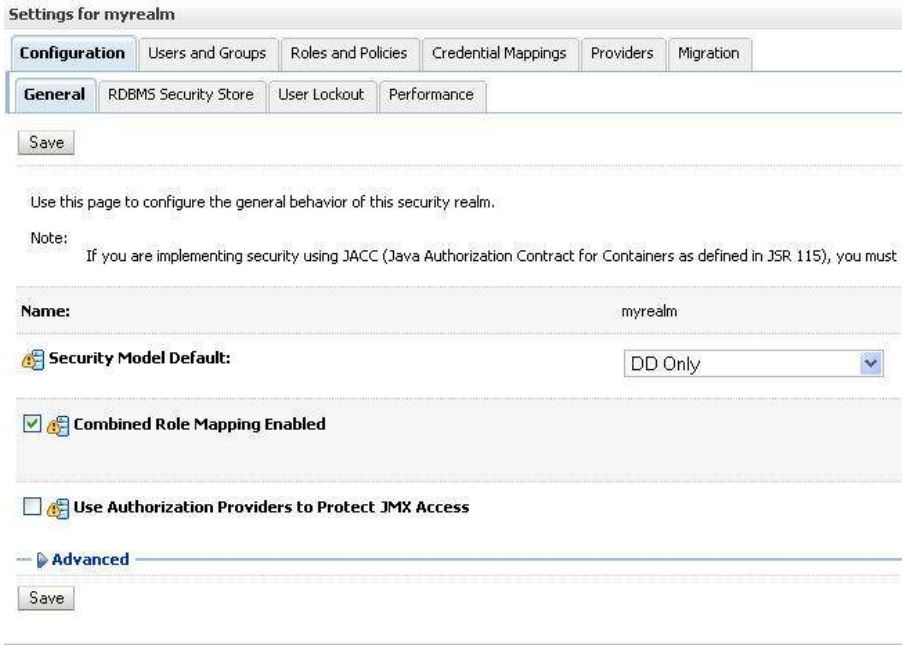
2. Click the **Security Realms** in the Domain Structure window.

   The *Summary of Security Realms* window opens, displaying the available realms. The default realm is **myrealm**.



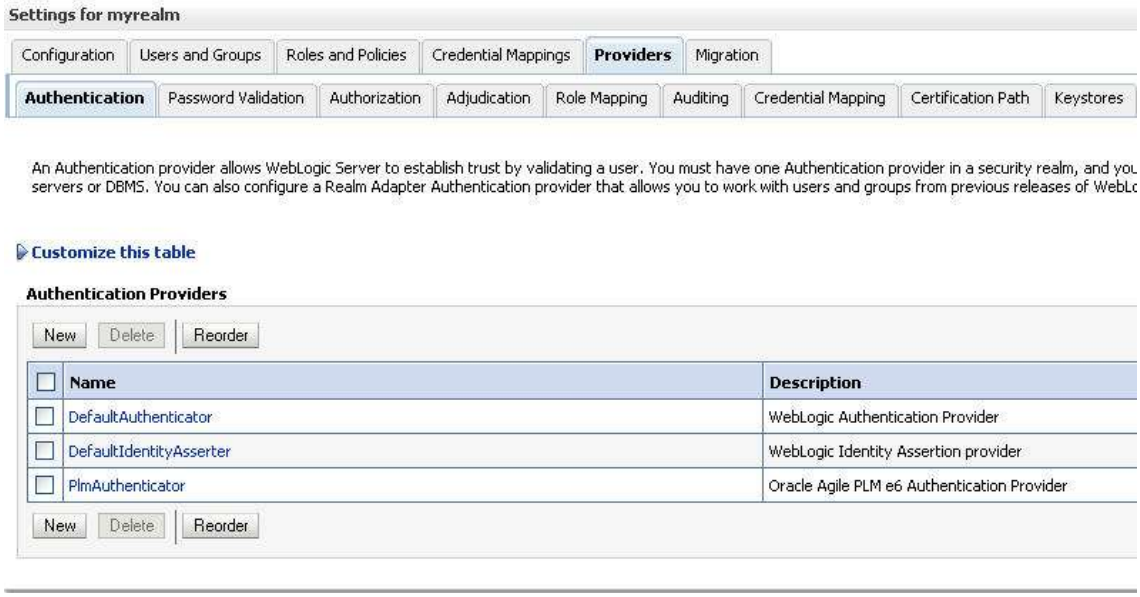
3. Click on **myrealm**.

   The *Settings for myrealm* window opens.



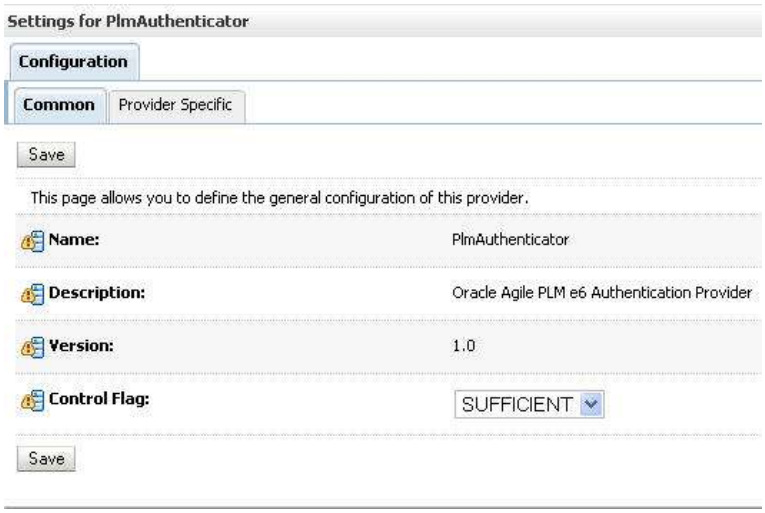
4. Go to the **Providers** tab.

   The PLM authentication providers are listed in the **Authentication** sub-tab.



5. Click **PlmAuthenticator** in the **Name** column.

   This opens the *Settings for PlmAuthenticator* window, displaying the Authentication **Configuration**.



6. In the **Common** sub-tab, select the **Control Flag** value.

   This value allows the configuration of the **JAAS (***Java Authentication and Authorization Service)* **login module** of the provider.

   The **JAAS Control Flag** values are:

REQUIRED	The Authentication provider is always called, and the user must always pass its authentication test. If authentication succeeds or fails, the authentication still continues down the list of providers.
REQUISITE	The user is required to pass the authentication test of the Authentication provider. If the user passes the authentication test of this Authentication provider, subsequent providers are executed but can fail (except for Authentication providers with the JAAS Control Flag set to REQUIRED).
SUFFICIENT	**[Recommended Value]** The user is not required to pass the authentication test of the Authentication provider. If authentication succeeds, no subsequent Authentication providers are executed. If authentication fails, authentication continues down the list of providers.
OPTIONAL	The user is allowed to pass or fail the authentication test of this Authentication provider. However, if all Authentication providers configured in a security realm have the JAAS Control Flag set to OPTIONAL, the user must pass the authentication test of one of the configured providers.

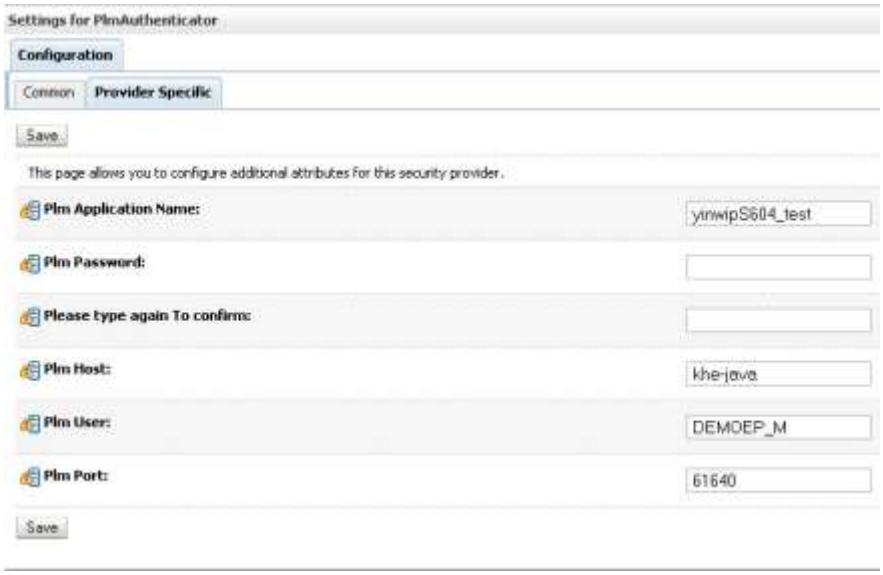
7. In the **Provider Specific** sub-tab, configure the following values:

   The PLM authenticator delegates the authentication request to the PLM server which is specified in the **Provider Specific** tab. This data are used by the PLM authenticator to connect to that PLM server.

   The user and password in the admin UI are used to configure the PLM authenticator. The PLM authenticator needs a valid PLM user and password from the PLM application belonging to its domain. It will start one e6 server and use this server then to check the user credentials in incoming HTTP(S) requests.

   The authenticator's main purpose is to have a fail-fast mechanism in case the HTTP(S) request contains wrong credentials. The authenticator just needs an ECI call to check the credentials. Without the authenticator, the web service session manager starts a new e6 server (this takes some time and valuable resources) just to notice that the credentials are incorrect.

It is OK to run a web service domain without authenticator, but you will have a performance penalty if many HTTP request with wrong credentials are coming in.



Plm Application Name	Name of PLM Application
Plm Password	PLM User password
Plm Host	Host name of the PLM Server
Plm User	Name of the PLM user
Plm Port	Port number of the Java Daemon for the PLM Server

Configuring the Authentication Provider for Web Services Hotfix Package

That is the only way right now by opening the WebLogic Admin console of the application domain.

1. Open **Security Realms**.
2. Open **myrealm**.
3. Open **PlmAuthenticator.**
4. Go to **Provider Specific** tab.

   Here you can set the **Plm Password** and **Plm User** values.

Note If you carry out the modifications for this application domain with Agile e6 Admin Client, the existing values are replaced with the default values that were used during the creation of this application domain.

You are required to repeat the steps given above each time you modify the application domain with the Agile e6 Admin Client or after you create a new application domain with the Agile e6 Admin Client where you need the Authenticator.

The values of *Plm Password* and *Plm User* are usually empty.

To configure the Authentication Provider in the case nothing was configured during Agile e6 installation, you have to use the WebLogic Administration Console for the application domain. It is only possible to configure the PLM User and Password without HF2 installed when the initial installation was done.

The Agile e6 Administration Client does not support the creation or modification of a new application where you can specify the user and password for this authenticator.

Testing the Inbound Web Services

Testing with the WebLogic Test Client

Once the Web Services Framework is set up and configured, you can test the functionality with the WebLogic Test Client.

With the WebLogic test client, you cannot call web service operations as you cannot pass the user credentials. However, it allows you to look at the WSDL and the schema files.

To map the calls without the user credentials, you can setup and use a Guest Account. See Using a Guest Account for more details.

The Guest Account should not be used in the production environment.

Warning	WebLogic test client does not work with the Web Services Security policies.

1. Log in to the Administration Console of the Oracle WebLogic Server

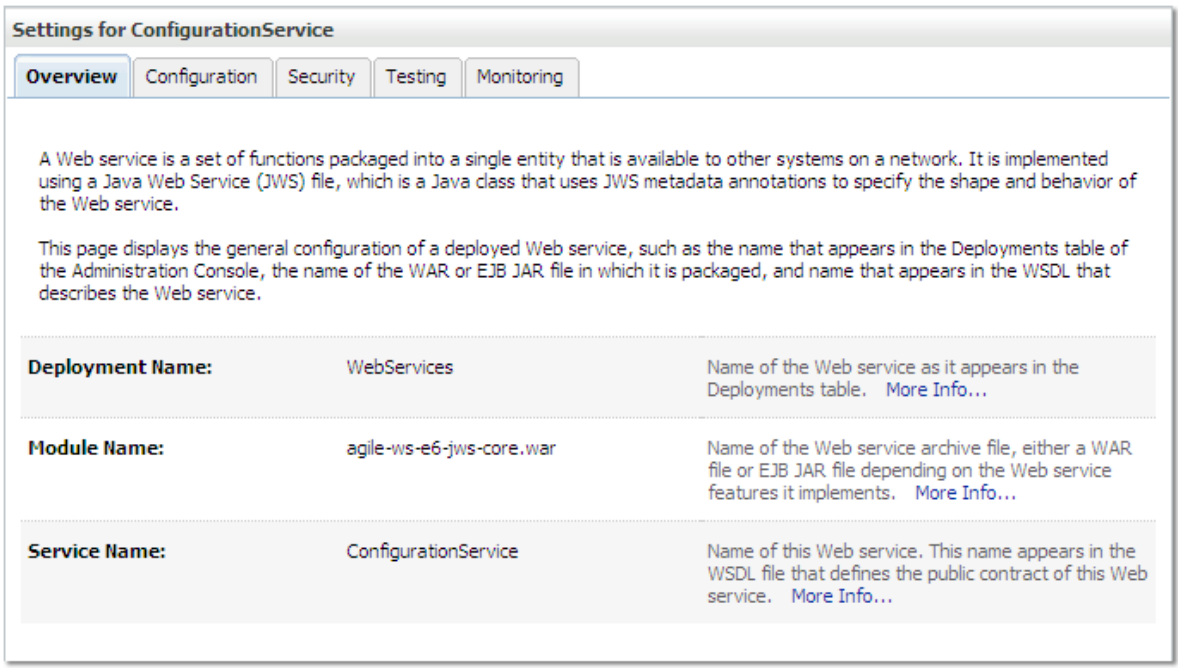
   For example, `http://server:7405/console/login/LoginForm.jsp`
2. In the left pane, click **Deployments** under the **Domain Structure.**

   The *Summary of Deployments* page appears displaying a table that lists out the **Deployments**.
3. In the Summary of Deployments list, click **WebServices**.

   The *Settings for MetadataService* page appears. By default, the Overview tab is open.
4. Scroll down to the **Modules and Components** list.
5. In the Modules and Components list, select the Web Service you wish to test.

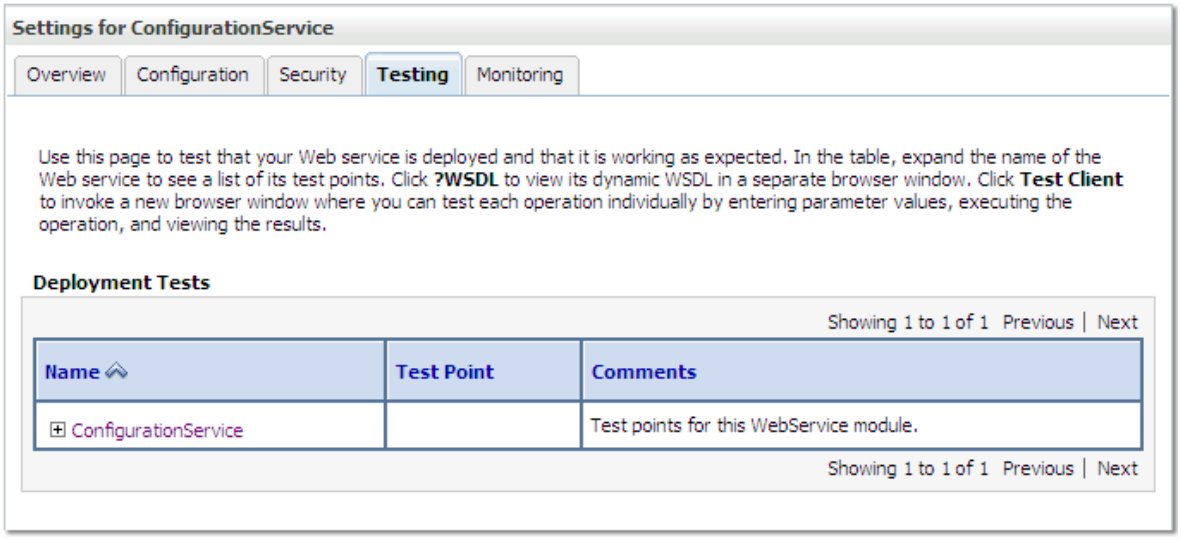
   In the following example, we will test the **ConfigurationService** Service.

   The *Settings for ConfigurationService* page appears.



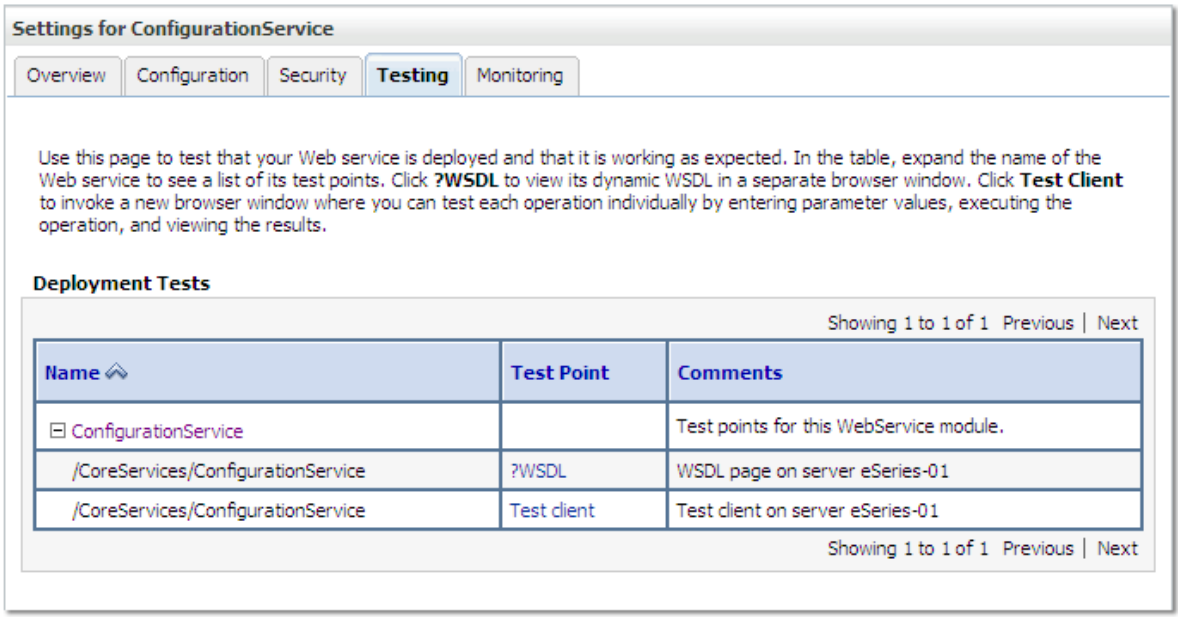
6. Go to the **Testing** tab.

   The *Deployment Tests* list appears.



7. Expand the entry **ConfigurationService.**

   The *Test Points* are displayed.



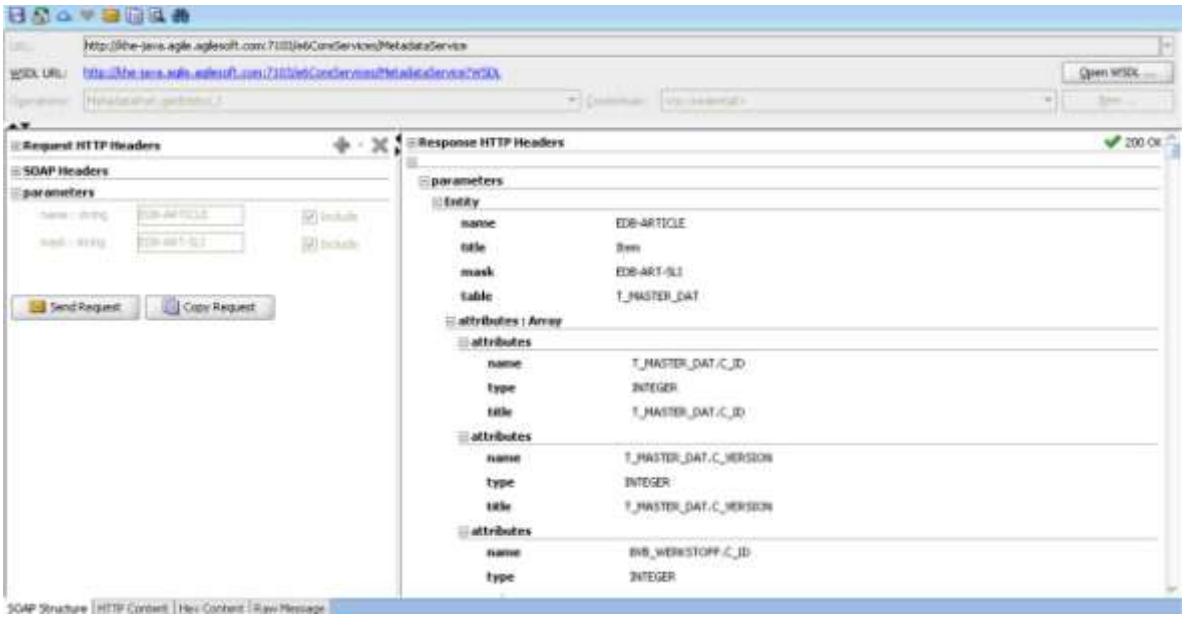
Testing with the JDeveloper HTTP Analyzer

The Web Service operations can also be called and tested using the JDeveloper HTTP Analyzer.

You are required to use HTTPS for the basic authentication with the HTTP analyzer otherwise you cannot call an operation. See the chapter Configuring the Agile e6 Web Services Security for more details.

To test the Web Services from the JDeveloper HTTP Analyzer:

1. Launch the JDeveloper IDE
1. Open the *HTTP Analyzer* screen
2. Enter the URL of the WSDL of the Web Service which you want to call.
3. In the *Request* area on the left, enter the input parameters for **name:string** and **mask:string,** such as EDB-ARTICLE and EDB-ART-SLI.
4. Click on **Send Request**.
5. Upon successful call of the web service, the result data is displayed in the Response area of the console.



Using a Guest Account

The PLM session manager supports a Guest Account feature which allows you to map the incoming requests without requiring the credentials to a PLM guest user.

This feature is needed to support the WebLogic Server test client, which does not support passing user credentials".

Warning	The Guest Account feature is not intended to be used in a production environment.

The guest account is configured by adding the following credentials into the **application.properties** file:

```
PlmUser=<plm.application.ws.guestuser>
PlmPwd=<plm.application.ws.guestpassword>
```

The password should be encrypted using the e6 installation encryption tool.

Note	Do not enter plain text passwords into configuration files.

The entry *PlmScope* tells the session manager if the Guest sessions should share an Agile e6 PLM server (`PlmScope=PUBLIC`) or if each Guest session should have its own Agile e6 server instance (`PlmScope=PRIVATE`).

Chapter 4

Configuring Agile e6 Web Services Security

The Web Services configuration can be used to configure the web services security. Since web services are not secured by default, it is required to configure/establish a web service policy that meets your security strategy requirements.

Note	The Agile e6 web service needs user credentials to connect to the Agile e6 application server. These credentials are provided by the Agile e6 authentication provider which stores this information into the Agile e6 principal. This Agile e6 principal is populated to the web service by the web service framework. If the web service is not configured to be secure, the Agile e6 principal will not be available for the Agile e6 web service. An insecure web service cannot work.
Note	The SSL port needs to be activated for the domain where the Web Services are deployed. The standard Listen port (non-SSL) should be disabled for the domain where the Web Services are deployed.

In the examples used in this chapter, a web service security policy is used to secure the entire web service. The client has to provide the WSS: SOAP message security user name token encrypted with a X.509 certificate.

Note	The certificate itself will also be authenticated. The certificate must be valid and the certificate name must be available in the system.

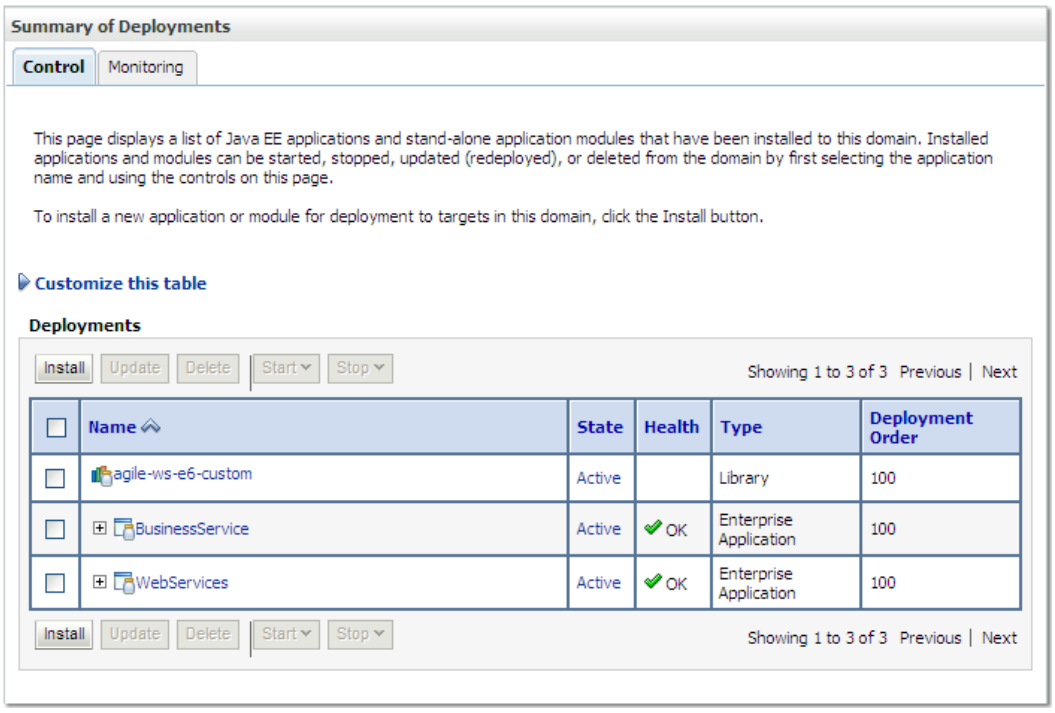
Setting up the Web Services Security Policies

Note	In the current release of Agile e6 core web services, *DocumentManagementService* does not support web services security policies. The file streaming only works with SSL (by using HTTPS). It does not work if HTTPS is enforced by adding a policy. All other web services can be controlled by the web services security policies.

To set up a Web Service Security Policy:

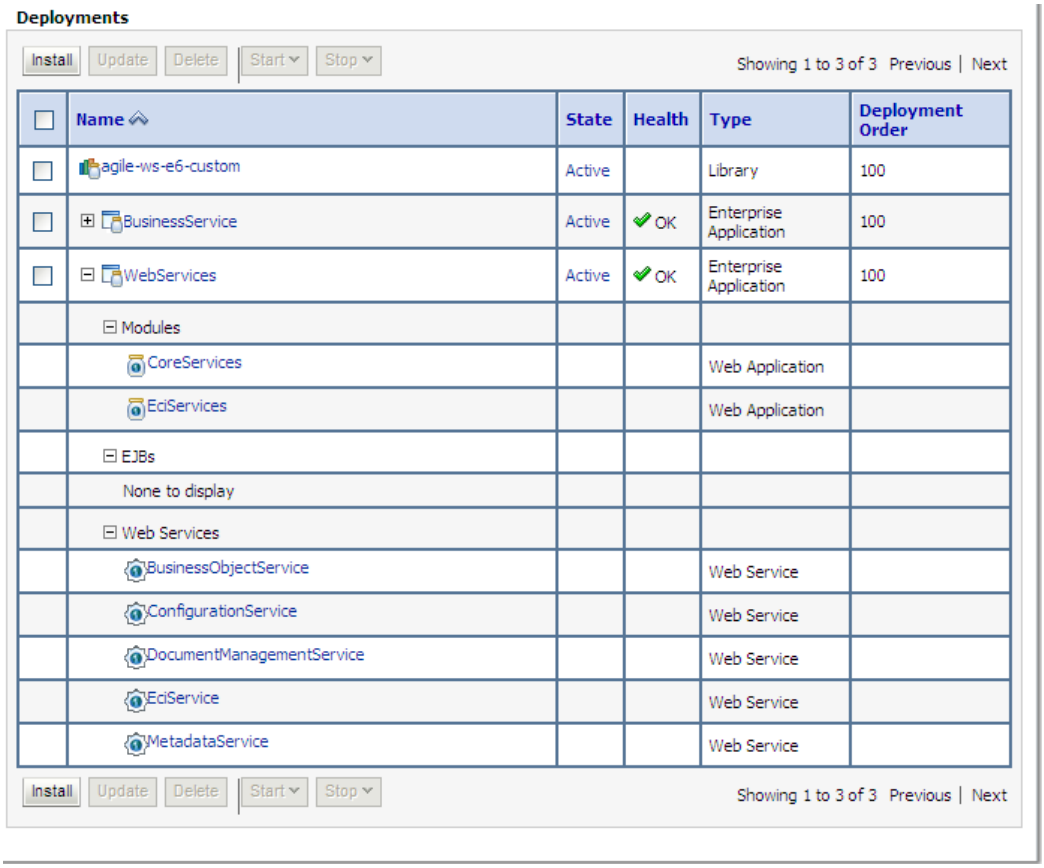
1. In the left pane, click **Deployments** under *Domain Structure*.

   The *Summary of Deployments* page appears with *Control* tab.



2. In the Name column, expand the **WebServices.**

   All the *Deployed Web Services* are listed.



3. Click a **Web Service**.

   For our further instructions, we will use **ConfigurationService**.

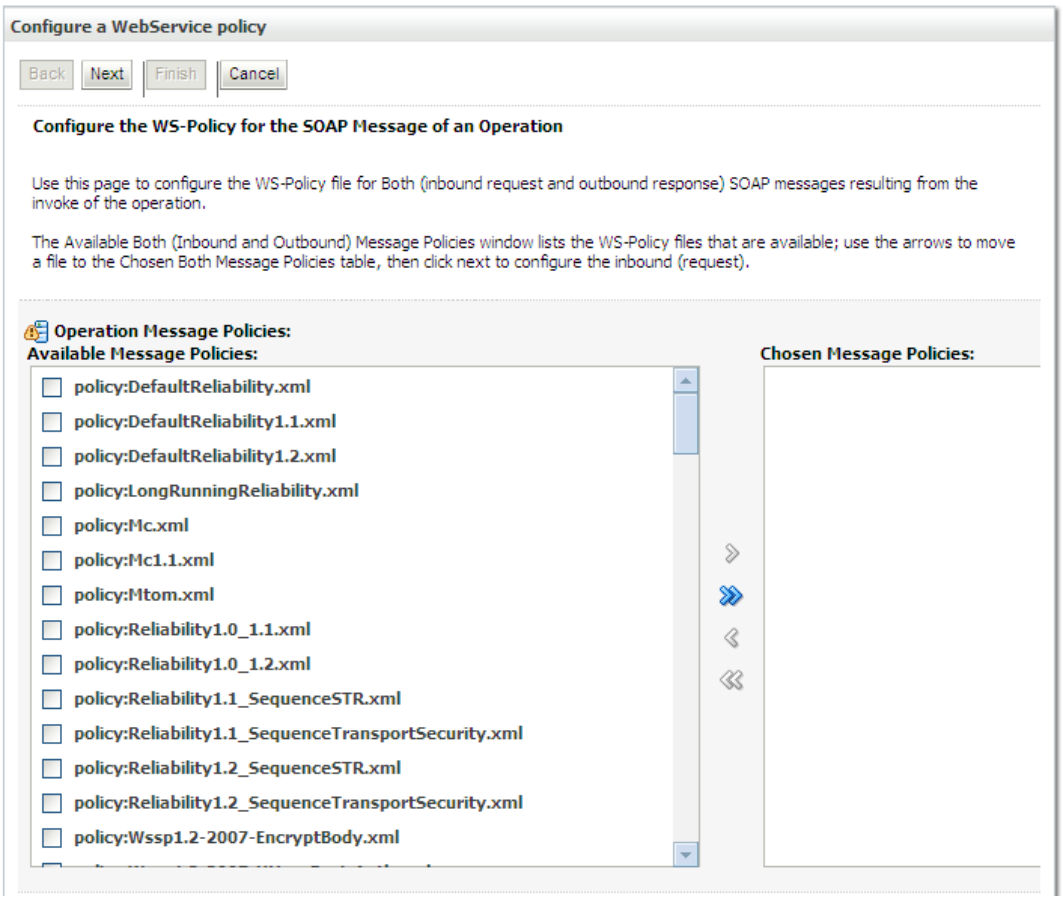
   The *Settings for ConfigurationService* page appears, displaying the *Service Endpoints and Operations* and corresponding *Policies*.

4. Go to the **WS-Policy** sub-tab in the **Configuration** tab and expand **ConfigurationPort** in *Service Endpoints and Operations*.

   The *WS-Policy* files associated with the ConfigurationService are listed.

5. Click an Operation, say **getDefault**.

   The *Configure a WebService Policy* page to **Configure the WS-Policy for the SOAP Message of an Operation** appears.



6. Select the desired **Operation Message Policies** from the **Available Message Policies** box and move them to the **Chosen Message Policies** box.
7. Click **Next**.

   The *Configure a WebService Policy* page to **Configure the WS-Policy for the Inbound SOAP Message of an Operation** appears.
8. Select the desired **Operation Message Policies** from the **Available Message Policies** box and move them to the **Chosen Inbound Message Policies** box.
9. Click **Next**.

   The *Configure a WebService Policy* page to **Configure the WS-Policy for the Outbound SOAP Message of an Operation** appears.
10. Select the desired **Operation Message Policies** from the **Available Message Policies** box and move them to the **Chosen Outbound Message Policies** box.
11. Click **Finish.**

    The *Save Deployment Plan Assistant* page appears.
12. Click **OK** to save the policies and return to the *Settings for ConfigurationServices* page.

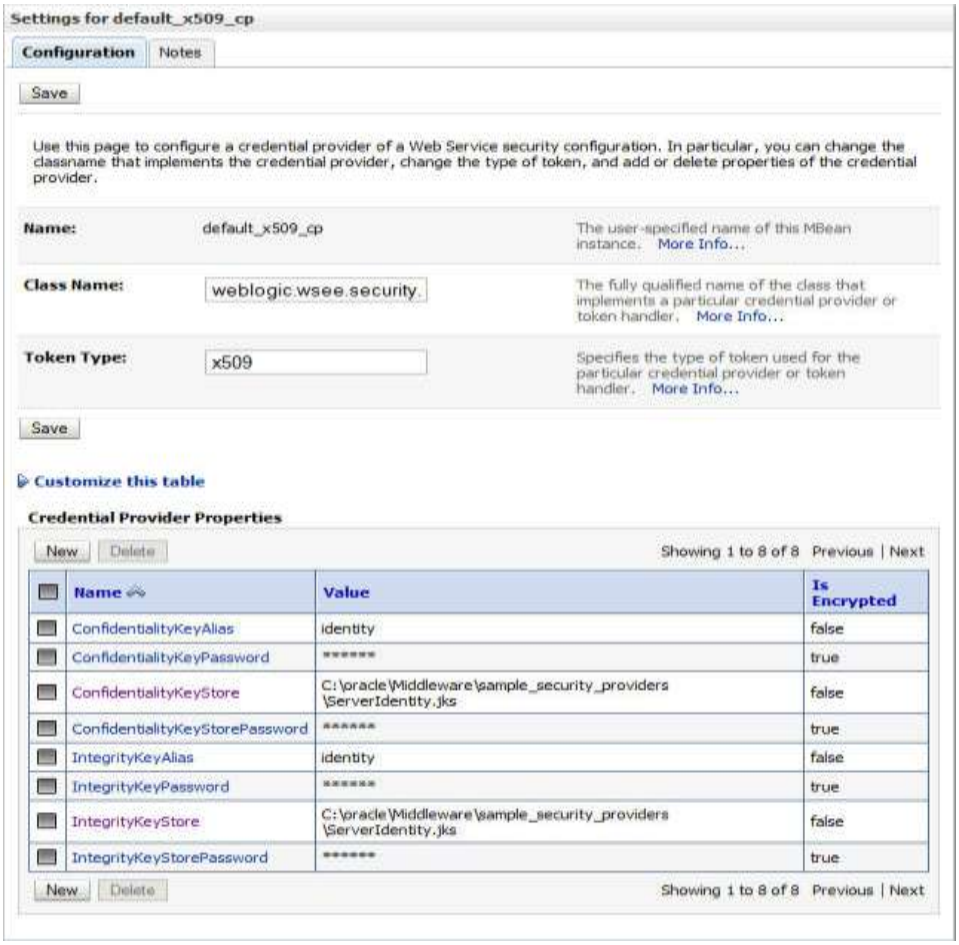
Setting up the Web Services Security

X.509 Authentication

For X.509 authentication, a **Web Service Security Configuration** must be configured.

The WebLogic server provides default providers which can be used. In this example, clicking on the default Web Service **default_wss** opens the ***Settings for default_wss*** mask. On the ***Credential Provider*** tab, a number of available default providers are listed that have been created for the web service security configuration.

The x509 server certificate has to be stored in a Java keystore and can be configured as follows:



Authenticating in a Web Service Client

The caller of an Agile e6 Web Service has to provide user credentials to gain access to the Agile e6 application. The attributes of these credentials depend on the used web service policy.

The following is an example of an unauthenticated Web Service call:

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns4:getVersion xmlns:ns2="http://xmlns.oracle.com/Agile/e6/Metadata/v0"
                        xmlns:ns3="http://xmlns.oracle.com/Agile/e6/plm"
xmlns:ns4="http://xmlns.oracle.com/Agile/e6/HelloWorld/v0" />
    </S:Body>
</S:Envelope>
```

A Sample of HTTP Authentication

You can use the basic authentication of HTTP to secure a web service. With this basic authentication of HTTP, the user credentials are stored in the HTTP header. The SOAP message does not carry any security information.

Basic authentication without SSL should not be used in a production environment as the passwords

and data are transferred in plain text. Ideally, HTTP/S should be configured.

For complete details on WebLogic Security Fundamentals and Transport Level Security, refer to the WebLogic documentation on OTN.

Here is a sample of basic HTTP authentication:

```
POST /agile-ws-e6/HelloWorldService HTTP/1.1
Authorization: Basic RURCQ1VTVE86T1RTVUNCREU=
SOAPAction: ""
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2,
*/*; q=.2
Content-Type: text/xml;charset="utf-8"
User-Agent: Oracle JAX-WS 2.1.4
Host: localhost:7002
Connection: keep-alive
Content-Length: 315

<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns4:getVersion xmlns:ns2="http://xmlns.oracle.com/Agile/e6/Metadata/v0"
                        xmlns:ns3="http://xmlns.oracle.com/Agile/e6/plm"
xmlns:ns4="http://xmlns.oracle.com/Agile/e6/HelloWorld/v0" />
    </S:Body>
</S:Envelope>
```

To add the HTTP basic authentication to a SOAP request, the code may look like the following example:

```
MetadataService service = new MetadataService(wsdlURL, serviceQName);

BindingProvider bindingProvider = (BindingProvider) service.getPort();

bindingProvider.getRequestContext().put(BindingProvider.USERNAME_PROPERTY,
username);
bindingProvider.getRequestContext().put(BindingProvider.PASSWORD_PROPERTY,
password);
bindingProvider.getRequestContext().put(BindingProvider.SESSION_MAINTAIN_PROPERTY,
Boolean.TRUE);
```

Note The last line in the example code given above configures HTTP session handling. If this code is not added, each web service call will create a new HTTP session which will lead to a new e6 server instance starting up.

A Sample of Web Services Security

In the following example, a **WS-Policy** (web services policy) is used for the **WSS: SOAP Message Security**.

You are required to provide the username token and a client certificate. The complete security information is embedded into the SOAP message.

Note By configuring WSS: SOAP Message Security, the WSDL gets modified.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header>
        <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd"
                       S:mustUnderstand="1">
```

```
            <ns1:EncryptedKey xmlns:ns1="http://www.w3.org/2001/04/xmlenc#"
Id="JOLZ6aDu8pt9PRPe">
                <ns1:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
                    <ns2:DigestMethod
xmlns:ns2="http://www.w3.org/2000/09/xmldsig#"

Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                </ns1:EncryptionMethod>
                <ns3:KeyInfo xmlns:ns3="http://www.w3.org/2000/09/xmldsig#">
                    <wsse:SecurityTokenReference xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
                                                 xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
                                                 wsu:Id="str_kNu7Vfo6pZLqdYcv">
                        <X509Data xmlns="http://www.w3.org/2000/09/xmldsig#">
                            <X509IssuerSerial>
                                <X509IssuerName>CN=CertGenCAB,OU=FOR TESTING
ONLY,O=MyOrganization,L=MyTown,ST=MyState,C=US</X509IssuerName>
                                <X509SerialNumber>-
135694037818432800534509206009756866711</X509SerialNumber>
                            </X509IssuerSerial>
                        </X509Data>
                    </wsse:SecurityTokenReference>
                </ns3:KeyInfo>
                <ns1:CipherData>

<ns1:CipherValue>JHUAfXjSBYxXKAGrpQ.............NUWGQ9IPL9M1uODqmnQ8Nlk=</ns1:Ciphe
rValue>
                </ns1:CipherData>
                <ns1:ReferenceList>
                    <ns1:DataReference URI="#PJr5jO7puKh5OL5b" />
                </ns1:ReferenceList>
            </ns1:EncryptedKey>
            <wsse:BinarySecurityToken xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
                                      xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
                                      EncodingType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
                                      ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"

wsu:Id="bst_GpDR1niRFucsZsbm">MIICKzCc.......KMuSA1XAQ==</wsse:BinarySecurityToken
>
            <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
                <dsig:SignedInfo>
                    <dsig:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                    <dsig:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
                    <dsig:Reference URI="#Timestamp_KKvWCLdlrCRB2SNF">
                        <dsig:Transforms>
                            <dsig:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                        </dsig:Transforms>
                        <dsig:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />

<dsig:DigestValue>xndjH7PWB/yinv/uFzmElQzAezI=</dsig:DigestValue>
                    </dsig:Reference>
                    <dsig:Reference URI="#Body_0lUdcO0zqWY2bBvU">
                        <dsig:Transforms>
                            <dsig:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                        </dsig:Transforms>
                        <dsig:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />

<dsig:DigestValue>gt0av56Xh/gca30jxtDChJkFZck=</dsig:DigestValue>
                    </dsig:Reference>
                    <dsig:Reference URI="#unt_UROJpKRFSAIZLKFf">
                        <dsig:Transforms>
                            <dsig:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
```

```
                        </dsig:Transforms>
                        <dsig:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />

<dsig:DigestValue>QBSH0z6BxmZgEM56+g3ZS2w00lg=</dsig:DigestValue>
                    </dsig:Reference>
                    <dsig:Reference URI="#bst_GpDR1niRFucsZsbm">
                        <dsig:Transforms>
                            <dsig:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                        </dsig:Transforms>
                        <dsig:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />

<dsig:DigestValue>62PwFQZD1Nj1R77qudrvzJCIUNE=</dsig:DigestValue>
                    </dsig:Reference>
                </dsig:SignedInfo>

<dsig:SignatureValue>TfFLyCR9MF4ZepqwmnCned7mj5TavfwjDg69.......MIFR3kBU=</dsig:Si
gnatureValue>
                <dsig:KeyInfo>
                    <wsse:SecurityTokenReference xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
                                                 xmlns:wsse11="http://docs.oasis-
open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd"
                                                 xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"

wsse11:TokenType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-
token-profile-1.0#X509v3"
                                                 wsu:Id="str_KVJy1AtKNAxVYsKq">
                        <wsse:Reference URI="#bst_GpDR1niRFucsZsbm"
                                        ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" />
                    </wsse:SecurityTokenReference>
                </dsig:KeyInfo>
            </dsig:Signature>
            <ns1:EncryptedData xmlns:ns1="http://www.w3.org/2001/04/xmlenc#"
Encoding="UTF-8" Id="PJr5jO7puKh5OL5b" MimeType="text/xml"
                               Type="http://www.w3.org/2001/04/xmlenc#Element">
                <ns1:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc" />
                <ns1:CipherData>

<ns1:CipherValue>yG4ULSKvJFL8......OgqkcPmY6yhdpoE=</ns1:CipherValue>
                </ns1:CipherData>
            </ns1:EncryptedData>
            <wsu:Timestamp xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
                           wsu:Id="Timestamp_KKvWCLdlrCRB2SNF">
                <wsu:Created>2010-02-03T14:44:31Z</wsu:Created>
                <wsu:Expires>2010-02-03T14:45:31Z</wsu:Expires>
            </wsu:Timestamp>
        </wsse:Security>
    </S:Header>
    <S:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
            wsu:Id="Body_0lUdcO0zqWY2bBvU">
        <ns4:hello xmlns:ns2="http://xmlns.oracle.com/Agile/e6/Metadata/v0"
                   xmlns:ns3="http://xmlns.oracle.com/Agile/e6/plm"
                   xmlns:ns4="http://xmlns.oracle.com/Agile/e6/HelloWorld/v0" />
    </S:Body>
</S:Envelope>
```

In the following example, **WSS: SOAP Message Security** information is provided for the SOAP request:

```
MetadataService service = new MetadataService(wsdlURL, serviceQName);
BindingProvider bindingProvider = (BindingProvider) service.getPort();
List<CredentialProvider> credProviders = new ArrayList<CredentialProvider>();
try {
  // Load server certificate
```

```
  X509Certificate serverCert =
(X509Certificate)CertUtils.getCertificate(serverCertFile);

  // Check server certificate
  serverCert.checkValidity();

  // Create a new certificate credential provider
  CredentialProvider cp = new ClientBSTCredentialProvider(clientKeyStore,
              clientKeyStorePass,  clientKeyAlias, clientKeyPass,  "JKS",
serverCert);

  // Add certificate credential provider to the array list
 credProviders.add(cp);

  // Create a new username token credential provider
  CredentialProvider up = new ClientUNTCredentialProvider(username.getBytes(),
password.getBytes());

  // Add certificate username token credential provider to the array list
  credProviders.add(up);

  Map<String, Object> rc = ((BindingProvider)portName).getRequestContext();

  // Add the credential providers to the request context
  rc.put(WSSecurityContext.CREDENTIAL_PROVIDER_LIST, credProviders);

  // Add a trusr manager to the request context, you can do here some validation
tests on the return certificate of the SOAP message
  rc.put(WSSecurityContext.TRUST_MANAGER,
                    new TrustManager(){
                      public boolean certificateCallback(X509Certificate[] chain,
int validateErr){
                        return true;
                      }
                    } );
} catch (Exception e) {
  log.printStackTrace(e);
}
```

Chapter 5

Working with Agile e6 Web Services

Bulk Processing of Requests

Most of the Agile e6 web service operations support bulk processing of requests. The operations with one request input object and one response output object can be configured to process multiple or bulk requests and corresponding responses.

Handling the Bulk Requests

A bulk request contains a list of requests for the non-bulk operation. These requests are executed one by one, and each response is stored in the result list of the bulk response object.

All requests contained in the bulk request list are executed in sequence using the order of the list.

The bulk requests can be configured to either stop on the first failure, or to continue regardless of a failing request.

If a request fails with a response FAILURE, the loop will be aborted if the *stopOnFailure* member of the bulk request is set. If *stopOnFailure* is not requested, the status of the bulk response is set to *PARTIAL_SUCCESS*. This requires you to look into each response in the list to check the individual status.

However, if a request fails with a *PlmFault* (or any other exception), the bulk processing is aborted and the list of requests processed until the fault occurred is returned. Additionally, the causing fault is returned in the bulk response.

If the bulk request does not contain any request, a WARNING response is returned.

The content of the response object depends on the status code, as listed below:

SUCCESS	A list with all response objects matching the list of requests. All requests succeeded.
FAILURE	A list with all response objects matching the list of requests. One or more requests failed, check the respective responses. The last executed request failed with an exception, which is returned as the fault in the bulk response. If one request fails due to lack of mask in White-list, status code would be FAILURE instead of PARTIAL_SUCCESS even if other services should are successful.
WARNING	The bulk request does not contain any request.
PARTIAL_SUCCESS	A list with all response objects matching the list of requests. One or more requests failed, check the respective responses. If *stopOnFailure* was requested, the list ends with the first response that failed.

Developing the Outbound Web Services Wrapper

The Web Services Wrapper Interface

Each wrapper has to implement the *WebServiceWrapper* interface, which prescribes the following method:

```
StringList callWebService(WrapperContext context, CallableParam args)
```

The context contains all relevant information for the wrapper to perform the call. The wrapper then transforms the arguments to an XML payload for the outbound web service and finally makes the call.

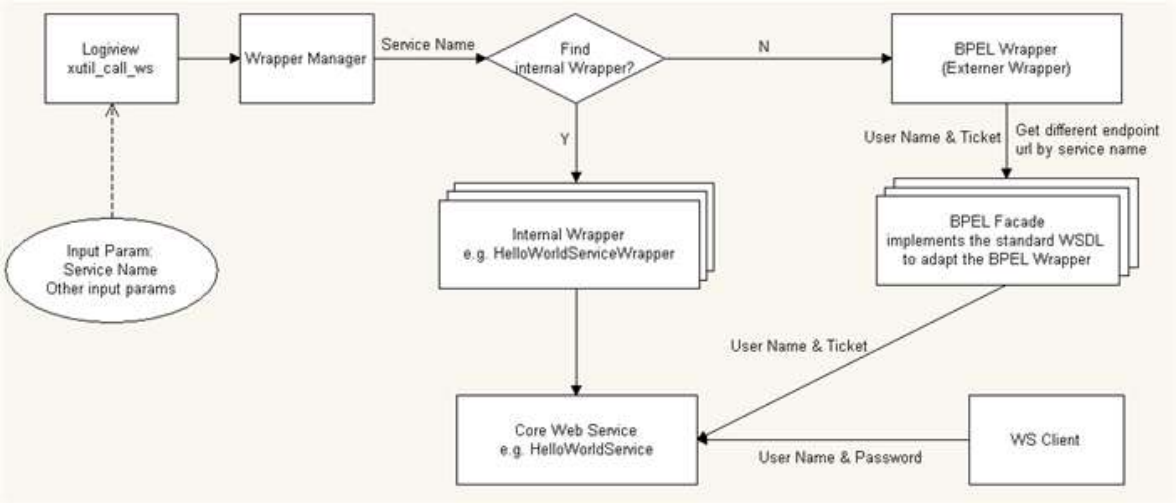
If it is an asynchronous web service, the wrapper returns a string list with the correlation ID. Otherwise, it transforms the XML payload returned by the web service into a string list that is expected by the calling Agile e6 server process.

In case the wrapper is implemented in BPEL (or in any other external web service language), you require a special wrapper called the *ExternalWrapper*. This wrapper delegates the call to the respective external web service wrapper.

The BPEL Facade

For the outbound calls, you are required to use the *ExternalWrapper* to call an external web service or a BPEL process. However, you need to first implement an interface to adapt the External Wrapper. This interface is called as *BPEL facade*.

Normally, this facade should be a BPEL process that you implement. In this facade, you can invoke an external web service or BPEL process and create their business logic. All the facades should implement a standard WSDL. The External wrapper uses this standard WSDL to generate the proxy. Then it can use different endpoints to call different BPEL facades.



Endpoint Configurations for the External Wrapper

All the endpoints for the External wrapper are defined in the properties file

ExternalWrapper.properties. This file resides in the *APP-INF/classes* directory alongside the *application.properties* file for the Web Services.

Example

```
bpel.wrapper.SampleExternalService2.wsdl=http://<server><port>/soa-
infra/services/default/SOAComposite2/BPELFacadeService?WSDL
```

In the above example, when we pass the service name as *SampleExternalService2* in **xutil_call_ws**, the wrapper manager first looks for an internal wrapper named as *SampleExternalService2*. If this internal wrapper does not exist, then the wrapper manager calls the external wrapper and gets the endpoint with the key value *bpel.wrapper.SampleExternalService2.wsdl* in the **ExternalWrapper.properties** file.

Note	You are required to use the custom staging mechanism of the Agile e6 Installer to add your own mappings to the *ExternalWrapper.properties* file.

Session Management Integration

In the *ExternalWrapper*, the user name & ticket should be passed to the BPEL façade. The BPEL facade uses this information to invoke the e6 core web services and reconnect to the same Agile e6 PLM session to get more data.

Developing a Custom Wrapper

In order to compile a custom wrapper, you need the libraries contained in the Web Services application. These libraries can be found at ${ep_root}/staging/product/WebServices/WebServices.ear/APP-INF/lib.

If your wrapper calls an external or internal web service, you also need to add the generated client classes or any other infrastructure classes needed. This depends on the web service client framework that you use.

Important	It is highly recommended to use the WebLogic Web Service framework when implementing a wrapper for a web service because the wrapper is runs inside WebLogic.

The name of the wrapper class should be **<ID>Wrapper,** as the wrapper manager looks for this string when the Agile e6 server tries to call it.

The imports used by the wrapper are in the libraries contained in the lib directory inside the *WebServices.ear* file (at APP-INF/lib). By default, the wrapper class should be in the package **com.agile.ws.e6.wrappers** so that the wrapper manager can find it at the runtime. However, it is also possible to add other package names to the search path by adding them to the **application.properties** file of the Web Services application:

```
WrapperPackage.Custom.1 = some.custom.wrapperpackage
WrapperPackage.Custom.2 = another.custom.wrapperpackage
```

Example: EchoServiceWrapper.java

This wrapper does not call an outbound web service. It returns the arguments that have been passed. This wrapper is already contained inside the Web Services application and can be used to test the infrastructure.

```
/*
 * $Id: EchoServiceWrapper.java,v 1.3 2010/10/15 15:11:56 brg Exp $
 *
 * Copyright (c) 1992, 2010, Oracle. All rights reserved.
```

```
 */
package com.agile.ws.e6.wrappers;

import com.agile.eci.EciConnection;
import com.agile.eci.EciPar;
import com.agile.eci.EciParBuffer;
import com.agile.share.callable.CallableParam;
import com.agile.share.trace.Logger;
import com.agile.share.trace.Trace;
import com.agile.share.util.StringArray;
import com.agile.share.util.StringList;
import com.agile.ws.e6.PlmSession;
import com.agile.ws.e6.wrappersupport.WrapperContext;

import java.net.InetAddress;

/**
 * A simple echo wrapper to test the wrapper mechanism.
 */
public class EchoServiceWrapper extends AbstractWrapper {

   private static Logger log = Trace.getLogger(EchoServiceWrapper.class);

   /**
    * The name of this service wrapper.
    */
   public static final String NAME = EchoServiceWrapper.class.getSimpleName();

   /**
    * Creates a new echo service.
    */
   public EchoServiceWrapper() {
      super(NAME);
   }

   /** {@inheritDoc} */
   @Override
   public StringList callWebService(WrapperContext context, CallableParam args) {

      log.enter("callWebService", "context="+context+", args="+args);
      PlmSession session = null;
      String userName = "<unknown>";
      String processId = "<unknown>";

      try {
         session = createPlmSession(context);
         EciConnection con = session.getConnection();

         EciPar par = con.call("eci_rea_edb_usr");

         userName = par.get(1);
         log.info("My name is " + userName + "," +
               "\nI belong to the group " + par.get(2) +
               ".\nMy user ID is " + par.get(3) + "," +
               "\nmy group ID is " + par.get(4) +
               "\nand I am " + ("y".equals(par.get(5)) ? "" : "not ") + "a
manager" +
               ".\n" +
               "\nOur session is " + session + ".\n");

         par = con.call("eci_get_pid");
         processId = par.get(1);

         EciParBuffer buf = new EciParBuffer();
         buf.add("EDB-BAS-WARNING");
         buf.addNew("This is the e6 EchoService running inside WebLogic on host "
+
               InetAddress.getLocalHost().getHostName() + "\n\n" +
               "You are " + userName + " and your process ID is " + processId);
         buf.end();
         con.call("eci_mes_wri", buf);
      }
      catch (Exception e) {
         log.error("Unable to reconnect to e6: ", e);
      }
```

```
      finally {
         if (session != null) {
            session.close();
         }
      }
      StringList result = new StringArray(args.getParam());
      log.leave("callWebService", "result=" + result);
      return result;
   }
}
```

Example: SampleWrapper.java

This wrapper calls an e6 core service as an example. The generated client code needed to call the e6 core service is not included.
The generated classes belong to the wrapper and need to be deployed along with it.

The *SampleWrapper* class needs a property file, *SampleWrapper.properties,* which contains the URL of the web service.

When Agile e6 application wants to call a wrapper called Sample, it will pass **Sample** as the first argument to **xutil_call_ws**. The wrapper manager will then look for a class called **SampleWrapper** in all the packages in its search path. To call the EchoServiceWrapper, pass **EchoService** to **xutil_call_ws**.

```
/*
 * $Id: SampleWrapper.java,v 1.1.2.2 2011/06/30 15:53:27 brg Exp $
 *
 * Copyright (c) 1992, 2011, Oracle. All rights reserved.
 */

package com.agile.ws.e6.wrappers;

import com.agile.eci.EciConnection;
import com.agile.eci.EciPar;
import com.agile.eci.EciParBuffer;
import com.agile.security.tickets.plm.PlmTicket;
import com.agile.share.callable.CallableException;
import com.agile.share.callable.CallableParam;
import com.agile.share.trace.Logger;
import com.agile.share.trace.Trace;
import com.agile.share.util.StringArray;
import com.agile.share.util.StringList;
import com.agile.ws.e6.PlmSession;
import com.agile.ws.e6.WebServiceEnum;
import com.agile.ws.e6.client.core.common.PlmUserContext;
import com.agile.ws.e6.client.core.common.PlmUserContextUserInfo;
import com.agile.ws.e6.client.core.configuration.Configuration;
import com.agile.ws.e6.client.core.configuration.ConfigurationService;
import com.agile.ws.e6.client.core.configuration.GetUserContextRequestType;
import com.agile.ws.e6.client.core.configuration.GetUserContextResponseType;
import com.agile.ws.e6.wrappersupport.WrapperContext;

import java.net.InetAddress;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.UnknownHostException;
import java.util.Properties;

import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;

/**
 * A simple example of a wrapper for an out-bound web service call.
 *
 * <p>It extends the abstract wrapper implementation, which is shipped
 * in the library agile-ws-e6-callables as part of the WebServices.ear file.
 *
 * <p>At the customer site, the new wrapper classes need to be put into the
 * agile-custom shared library, which is referenced by the WebServices
application.
```

```
 *
 * <p>Alternatively, a JAR file containing the wrapper can be added to the
 * WebServices application into the APP_INF/lib directory,
 * so that it is deployed as part of the WebServices application.
 */
public class SampleWrapper extends AbstractWrapper {
   /** The Logger used to print trace messages */
   private static Logger log = Trace.getLogger(SampleWrapper.class);

   /** The name of our properties file */
   private static final String SAMPLE_PROPERTIES = "SampleWrapper.properties";
   /** Property containing the URL of the external web service */
   private static final String PROP_ENDPOINT = "Sample.endPoint";

   /**
    * The service we want to contact.
    *
    * <p>This class - and all the others in the package
com.agile.ws.e6.client.core - is
    * generated by the WebLogic clientgen Ant task.
    */
   private ConfigurationService configurationService;
   /** The URL to the Configuration service */
   private String configurationEndPoint;

   /**
    * The name of this service wrapper.
    */
   public static final String NAME = "Sample";

   /**
    * Creates a new HelloWorld service.
    */
   public SampleWrapper() {
      super(NAME);
   }

   /**
    * @return the RCS information of this object's class (polymorphic)
    */
   @Override
   public final String getRCSId() {
      return getClassRCSId();
   }

   /**
    * @return the RCS information of this class (static)
    */
   public static String getClassRCSId() {
      return "$Id: SampleWrapper.java,v 1.1.2.2 2011/06/30 15:53:27 brg Exp $";
   }

   /**
    * Creates the port to access the Configuration service.
    *
    * @param endPoint the WSDL URL of the service.
    * @param ticket the PLM ticket provided by the e6 server
    *
    * @return the port to call the Configuration service
    */
   private Configuration getConfigurationPort(PlmTicket ticket) throws
MalformedURLException {
      if (configurationService == null) {
         configurationService = new ConfigurationService(
               new URL(configurationEndPoint),
               new QName(WebServiceEnum.CONFIGURATION.getNamespace(),
                         WebServiceEnum.CONFIGURATION.getServiceName()));
      }

      Configuration port = configurationService.getConfigurationPort();

      BindingProvider binding = (BindingProvider) port;

      // Add authentication, we have a ticket so we do not need the password
```

```
      binding.getRequestContext().put(BindingProvider.USERNAME_PROPERTY,
ticket.getUserName());
      binding.getRequestContext().put(BindingProvider.PASSWORD_PROPERTY,
String.valueOf(ticket.getRawTicket()));

      // Maintain the HTTP session, in case we do several calls to the server
      binding.getRequestContext().put(BindingProvider.SESSION_MAINTAIN_PROPERTY,
Boolean.TRUE);

      return port;
   }

   /**
    * Calls the web service.
    *
    * @param context the context that contains the reference to the e6 server
    * @param args arguments passed by the e6 server
    * @return list of return values, or null
    *
    * @throws CallableException Web service call failed
    */
   @Override
   public final StringList callWebService(final WrapperContext context, final
CallableParam args) throws CallableException {
      log.enter("callWebService", "context="+context+", args="+args);

      // Try to load our properties
      Properties sampleProps = new Properties();

      try {
         sampleProps.load(getClass().getResourceAsStream("/" +
SAMPLE_PROPERTIES));
         configurationEndPoint = sampleProps.getProperty(PROP_ENDPOINT);
      }
      catch (Exception e) {
         String msg = "Unable to load SampleWrapper.properties";
         log.error(msg, e);
         throw new CallableException(this, msg);
      }
      if (configurationEndPoint == null) {
         log.error("No WSDL URL found in " + SAMPLE_PROPERTIES);
         throw new CallableException(this, "No web service URL configured for the
Sample wrapper");
      }

      log.info("Using WSDL at " + configurationEndPoint);

      Configuration port = null;
      StringList result = new StringArray();

      try {
         // Create a session object:
         //
         // It gives us access to an ECI connection to the same e6 server instance
that called us,
         // and - if needed - it will create an AxalantRepository instance for us,
         // if we want to make use of the high level Java ECI (JET layer).
         PlmSession session = createPlmSession(context);

         // First some ECI calls
         callEciDemo(session);

         // Now the "external" web service call.
         //
         // Here, we would normally call an external service of another system,
         // but for demo purposes, we just call an e6 Core service.
         port = getConfigurationPort(context.getPlmTicket());
         log.info("Port created with PLM ticket");

         GetUserContextRequestType request = new GetUserContextRequestType();
         GetUserContextResponseType response = port.getUserContext(request);
         log.info("Web method getUserContext() returned with status code " +
response.getStatusCode());

         PlmUserContext e6Context = response.getPlmUserContext();
```

```
         PlmUserContextUserInfo userInfo = e6Context.getPlmUserInfo();

         result.add("User name  = " + userInfo.getUserName());
         result.add("Group name = " + userInfo.getGroup());
         result.add("Language   = " + userInfo.getUserLanguage());
         result.add("Locale     = " + userInfo.getUserLocale());

      }
      catch (Exception e) {
         log.error("Unable to call web service", e);
         throw new CallableException(this, "Web service call failed", e);
      }
      finally {
         if (port != null) {
            try {
               // Tell the server that we no longer need the e6 session.
               port.closeSession();
            }
            catch (Exception e) {
               log.error("Unable to close port", e);
            }
         }
      }
      return result;
   }

   /**
    * Calls some ECI functions to demonstrate how to get additional data.
    *
    * @param session the e6 PLM session
    */
   private void callEciDemo(PlmSession session) {
      String userName = null;
      String processId = null;
      String host;

      try {
         host = InetAddress.getLocalHost().getHostName();
      }
      catch (UnknownHostException e) {
         host = "unknown host";
      }

      EciConnection con = session.getConnection();

      EciPar par = con.call("eci_rea_edb_usr");

      userName = par.get(1);
      log.info("My name is " + userName + "," +
            "\nI belong to the group " + par.get(2) +
            ".\nMy user ID is " + par.get(3) + "," +
            "\nmy group ID is " + par.get(4) +
            "\nand I am " + ("y".equals(par.get(5)) ? "" : "not ") + "a manager" +
            ".\n" +
            "\nOur session is " + session + ".\n");

      par = con.call("eci_get_pid");
      processId = par.get(1);

      EciParBuffer buf = new EciParBuffer();
      buf.add("EDB-BAS-WARNING");
      buf.addNew("This is the e6 SampleWrapper running inside WebLogic on host " +
            host + "\n\n" +
            "You are " + userName + " and your process ID is " + processId);
      buf.end();
      // Prints a message on the client that contacted us
      con.call("eci_mes_wri", buf);
   }
}
```

Calling a Custom Wrapper from e6

A new C/C++ user-exit is needed that can be called from LogiView or C/C++ to make an outbound web service call. The request is sent as an ECI call to the ECI server embedded in the application server, which then calls the respective wrapper for XML processing.

To limit any XML parsing in LogiView and C/C++, the user-exit sends a string containing a user-exit parameter as an input to the wrapper and it expects a list of string as a result from the wrapper.

The user-exit accepts an input argument that uses the syntax prescribed by *CallableParam* and *zag_cnv_arg*. This argument is passed to the ECI callable *Eci_call WebService* in the application server.

This user-exit is called **xutil_call_ws** and can be used in LogiView as follows:

```
EP_APP_CMD = "EchoService"
EP_APP_CID_STRING = ""
EP_APP_CONTENTS = "Please echo: something"
RES = #xutil_call_ws(EP_APP_CMD, EP_APP_CID_STRING, EP_APP_CONTENTS,
EP_APP_RESULT, EP_APP_ERROR)
     if (RES == 0)
          put(strprint("Web service %s returned: %s", EP_APP_CMD,
     EP_APP_RESULT))
     else
          put(strprint("Error %d when calling web service %s, error
     message is:\n%s", RES, EP_APP_CMD, EP_APP_ERROR))
     endif
```

This LogiView code uses four existing string variables and two new ones to call the wrapper for the *EchoService*. The EchoService returns the input arguments as a result, which is provided as a standard wrapper to test the infrastructure.

The user-exit **xutil_call_ws** consists of the following parameters:

EP_APP_CMD	Wrapper Name
EP_APP_CID_STRING	Correlation ID. This is used to correlate a later response, which may come in asynchronously, to the initiating request.
EPP_APP_CONTENTS	Any string value that the wrapper should interpret.
RES	Result string from the Wrapper
EP_APP_RESULT	A new string variable that needs to be created to call the wrapper for the EchoService.
EP_APP_ERROR	A new string variable that needs to be created to call the wrapper for the EchoService.
The user-exit provides a return code - “0” for success and any other number for errors.	

Deploying a Custom Wrapper

1. Identify the classes you need for your wrapper and create a JAR file containing them. Give your JAR files proper names to avoid naming conflicts with existing wrapper implementations.

   For example, use a unique prefix that identifies your company.

2. If you need third party JAR files, ensure that they are not already provided by WebLogic server or the e6 Web Services application. Do not use JAR files that duplicate features or come in conflict with the WebLogic server, like using different web service client frameworks.

   Be aware that your wrapper implementation will run inside and as part of a Web Service application deployed into WebLogic server.

3. Copy all JAR files needed by your wrapper implementation to the custom staging directory for the Web Services application. This should be:

   ```
   ${ep_root}/staging/custom/WebServices/${app_name}.ear/APP-INF/lib.
   ```

   Note Do not copy any library that is already part of the Web Services application into the custom staging area. These libraries are only replaced by updates from Oracle, and are always put in the product staging area.

4. Redeploy the Web Services into the WebLogic server.

See the *Agile e6 Administration Manual* or the *Hotfix Readme* for complete details on how to deploy an application.

Web Service Wrapper Log Messages

A log file look similar to the one shown below, depending on what has been implemented in the Java based Wrapper for message logging.

Chapter 6

Agile e6 Core Web Services Operations

This section describes the Agile e6 Core Web Services Operations. The use of these operations to process the bulk requests is described in section Bulk Processing of Requests.

Some of the operations, such as *getRelations*, require counting the number of records. For more information on these, refer to Counting the Objects in this document.

For additional information on these web services and operations, download the following from Oracle eDelivery Web Site edelivery.oracle.com:

- Agile e6 Web Services Schema Docs. Includes the PLM Data Types document.
- Agile e6 Web Services SOAP Samples

Business Object Web Services

The *Business Object Web Services* enable you to create and retrieve Agile e6 PLM objects belonging to an entity, an entity type and a relation.

All operations require one *Request* object as input and return one *Response* object.

- The request contains attribute values used to search or create a PLM object.
- The response contains the data of the respective objects.
- *Bulk operations* allow you to execute a whole list of requests with a single web service call. The response of a bulk operation contains a list of responses matching the list of requests.

The PLM objects are read from the Agile e6 application using the mask specified in the request. It is only possible to access PLM attributes that are visible in this mask, with the exception of the ID fields `EDB_ID` and `C_ID`. Only the masks listed in the *Web Service White List* of the Agile e6 application can be accessed.

- Use the `EBD_ID` if you need to keep the reference to a PLM object, especially if it is stored in another system.
- The `C_ID` should only be used to build internal object graphs, for instance when filling a UI element.

Note Not all PLM objects have an `EDB_ID` or a `C_ID`. Check the customization of the respective Agile e6 system before deciding how to make external references to PLM objects of that system.

All operations allow you to specify the attributes that you require to return from the PLM object. If no return attributes are specified, all the accessible attributes are returned, which corresponds to all visible fields of the mask. Optionally, all languages of multi-language attributes can be returned. Be aware that this may result in a considerably larger response object.

Binary Data Transfer

It is possible to request the transfer of binary data (BLOBs), however, by default, binary data is not transmitted. Be aware that the binary attributes to transfer need to be visible in the underlying Agile e6 mask. BLOBs are transferred using the MTOM feature of JAX-WS.

Bulk Operations

The following are the bulk operation names of the single request operations for Business Object web services described the concurrent sections. See Processing the Bulk Requests for complete details on how to handle the bulk operations.

- createObjectBulk
- createRelationBulk
- getObjectsBulk
- getRelationsBulk

createObject

Service To create a new PLM object in the e6 application.

Usage The request object contains the PLM Class name and an optional list of attribute names. On success, the response object returns the newly created PLM object name including all its attributes defined by the PLM.

Only existing and visible attributes are returned.

If the object already exists in the PLM or an error occurs during the creation of the object, an error code will be returned.

Request Type CreateObjectRequestType

- messageId (String): ID to be returned in the response (optional)
- messageName (String) : Name to be returned in the response (optional)
- plmObject: The PLM class reference and a list of object attributes (name/value pairs) used to create the new object.
  The meta data of the plmObject is currently ignored by this operation, so you can pass empty values for these elements.
  Only the list of attribute values is used to fill the new record in e6.
- plmResult: This describes how the result of the operation is returned to the client.
  - attributeNames (Boolean) : List of field names to be returned - (optional, default is all visible fields and ID fields).
  - includeBinaryValues (Boolean) : Include binary values? (optional, default is false)
  - includeAllLanguages (Boolean) : Include all languages? (optional, default is false)

Response Type CreateObjectResponseType

- statusCode (ResponseStatusCode) :
  - SUCCESS: the new created PLM object including all attributesdefined by the PLM result of the request
  - PARTIAL_SUCCESS: the new created PLM object including some of the attributes defined by the PlmResult of the request
  - FAILURE: a fault description including error message and kind of error.
- ticket (String) : A PLM ticket.
  The ticket can be used instead of the password in subsequent calls. It allows the client to reference the PLM session, even if the HTTP session has been closed, or - in case of a backward flow - if there is no HTTP session.
- The ticket is only valid for the PLM server instance that generated it.
- messageId (String): copied from the request, or generated.
- messageName (String): copied from the request, or the operation name.
- warnings: List of warnings (PlmWarningType) that occurred during the operation.
- exceptions: List of exceptions (PlmExceptionType) that occurred during the operation.
- object (PlmObject): the object created by this operation.
- It contains the attributes requested in the PlmResult, or all visible attributes if no attributes have been requested.

getObject

Service To retrieve the requested PLM objects from Agile e6 application.

Usage It executes a query on the Agile e6 application and returns the matching objects. The operation will use the mask that is specified in the request to retrieve the data (search in mask).

Response information can be restricted by defining the list of fields that should be returned. Be aware that you can get the values only for fields that are available in the mask.

The sorting of data determines the settings of the mask.

Date values are returned in UTC based on the assumption that the data returned by the PLM server is in Server Local Time.

By default, only the current language value will be retrieved for multi-language fields. However the web service client can request to retrieve all language values in the same call. The request member *plmQuery* controls this behavior, along with other settings that need to be described.

PlmQuery extends *PlmResult*, which provides the flag "*includeAllLanguages* (boolean, optional)". This flag indicates if all languages of multi-language fields should be returned, or just the current language of the session.

The *countOnly* flag of the request is very important here. For more information on this, refer to Counting the Objects in this document.

Request Type GetObjectsRequestType

- messageId (String): ID to be returned in the response (optional)
- messageName (String) : Name to be returned in the response (optional)
- plmQuery (PlmQuery):
  - plmClass: The object type as a PlmClassRef (PLM class reference, as provided by the Metadata service).
  - selection: A list of PlmCondition objects representing the search criteria for object attributes.
  - ignoreRecordLimit (Boolean): Ignore the record limit? - (optional, default is false)
  - countOnly (Boolean): Count only? - (optional, default is false))
  - attributeNames (Boolean) : List of field names to be returned - (optional, default is all visible fields and ID fields).
  - includeBinaryValues (Boolean) : Include binary values? (optional, default is false)
  - includeAllLanguages (Boolean) : Include all languages? (optional, default is false)

Response Type GetObjectsResponseType

- statusCode (ResponseStatusCode) :
  - Status SUCCESS: the query was executed without any problem. The list of objects might be empty.
  - Status PARTIAL_SUCCESS: one or more attributes, defined by the PlmResult of the request, were not found in the queried objects. Details can be found in the list of warnings (see below)
  - Status FAILURE: a fault description including error message and kind of error.

- ticket (String) : A PLM ticket.
  The ticket can be used instead of the password in subsequent calls. It allows the client to reference the PLM session, even if the HTTP session has been closed, or - in case of a backward flow - if there is no HTTP session.
  The ticket is only valid for the PLM server instance that generated it.
- messageId (String): copied from the request, or generated.
- messageName (String): copied from the request, or the operation name.
- warnings: List of warnings (PlmWarningType) that occurred during the operation.
- exceptions: List of exceptions (PlmExceptionType) that occurred during the operation.
- recordLimitHit (Boolean) : Indicates whether the query result has hit the mask limit.
- objects: List of objects (PlmObject) found including all the object attribute values listed in the PlmResult of the request
  If a count request was made, the response will have one PlmObject with an Integer attribute named COUNT containing the number of objects matching the query, and an Integer attribute called RECORD_LIMIT containing the current record limit of the mask used for the count operation.

createRelation

Service To create a relation between two PLM objects.

Usage It returns the data of the new relation object. The request object queries for the parent object, the query for the child object and the metadata that defines the relation type. The request must contain the object type, i.e. a PLM class reference.

Request Type CreateRelationRequestType

- messageId (String): ID to be returned in the response (optional)
- messageName (String) : Name to be returned in the response (optional)
- relationObject (PlmObject): the relation Object with the PLM class reference and a list of relationship attributes (name/value pairs) used to create the new object. .
  The meta data of the relationObject is currently ignored by this operation, so you can pass empty values for these elements.
  Only the list of attribute values is used to fill the new relation record in e6.
- parent (PlmObjectRef): object reference : This is either a PlmObject as returned by the BusinessService.getObjects operation, or a PlmObjectReference containing the query for the record,
- child (PlmObjectRef): as object reference : This is either a PlmObject as returned by the BusinessService.getObjects operation, or a PlmObjectReference containing the query for the record,
- relation (PlmMetaRelation): a relationship meta data: This is a PlmMetaRelation as returned by the MetadataService.getRelation operation. It defines the relation of the parent and the child object.
- plmResult (PlmResult):This describes how the result of the operation is returned to the client.
  - attributeNames (Boolean) : List of field names to be returned - (optional, default is all visible fields and ID fields).
  - includeBinaryValues (Boolean) : Include binary values? (optional, default is false)
  - includeAllLanguages (Boolean) : Include all languages? (optional, default is false)

Response Type CreateRelationResponseType

- statusCode (ResponseStatusCode) :
  - SUCCESS: the new created PLM object including all attributes defined by the PlmResult of the request
  - PARTIAL_SUCCESS: the new created PLM object including some of the attributes defined by the PlmResult of the request
  - FAILURE: a fault description including error message and kind of error.
- ticket (String) : A PLM ticket.
  The ticket can be used instead of the password in subsequent calls. It allows the client to reference the PLM session, even if the HTTP session has been closed, or - in case of a backward flow - if there is no HTTP session.
  The ticket is only valid for the PLM server instance that generated it.
- messageId (String): copied from the request, or generated.
- messageName (String): copied from the request, or the operation name.
- warnings: List of warnings (PlmWarningType) that occurred during the operation.
- exceptions: List of exceptions (PlmExceptionType) that occurred during the operation.

- relationObject (PlmObject): The created PLM relation object contains the attributes listed in the PlmResult of the request.
  Additionally the ID attributes for of the parent and the child record are add. The parent IDs are named PARENT.EDB_ID and PARENT.C_ID, the attributes for child IDs are named CHILD.EDB_ID and CHILD.C_ID. This is necessary to prevent name clashes if the parent and child entity are the same (for instance in a item BOM).

getRelations

Service To get the relationship between two or more PLM objects.

Usage The operation executes a query on the Agile e6 application and returns the matching PLM objects. The operation uses the mask that is specified in the relation to retrieve the data (search in mask).

By default the response returns values of all visible fields (plus some important ID fields like EDB_ID and C_ID) that are contained in the mask. Response information can be restricted by defining the list of fields that should be returned, but be aware that you can only return values for fields which are available in the mask. The settings of the mask will determine the sorting of the data. Return values will be provided in a standardized format as marshaled by the JAXB framework. Date values will be in UTC, based on the assumption that the data returned by the PLM server is in server local time.

By default, only the current language value will be retrieved for multi-language fields. However the web service client can request to retrieve all language values in the same call.

Objects for which the current user does not have access will not show up in the response.

The *countOnly* flag of the request is very important here. For more information on this, refer to Counting the Objects in this document.

Request Type GetRelationsRequestType

- messageId (String): ID to be returned in the response (optional)
- messageName (String) : Name to be returned in the response (optional)
- plmRelationQuery (PlmRelationQuery):
  - PlmObject or PlmObjectReference: To get the parent record of the relation. (PlmObject as returned by the BusinessObjectService)
  - PlmMetaRelationRef: The object type is a PLM relation reference (parent, child, type, view).
  - selection: A list of PlmCondition objects representing the search criteria for object attributes.
  - ignoreRecordLimit (Boolean): Ignore the record limit? - (optional, default is false)
  - countOnly (Boolean): Count only? - (optional, default is false)
  - attributeNames (Boolean) : List of field names to be returned - (optional, default is all visible fields and ID fields).
  - includeBinaryValues (Boolean) : Include binary values? (optional, default is false)
  - includeAllLanguages (Boolean) : Include all languages? (optional, default is false)

Response Type GetRelationsResponseType

- Status code (SUCCESS, PARTIAL_SUCCESS, WARNING or FAILURE).
- A PLM ticket (String)
  The ticket can be used instead of the password in subsequent calls. It allows the client to reference the PLM session, even if the HTTP session has been closed, or - in case of a backward flow - if there is no HTTP session.
  The ticket is only valid for the PLM server instance that generated it.
- Message ID (copied from the request, or generated).

- Message name (copied from the request, or the operation name).
- List of warnings (PlmWarningType) that occurred during the operation.
- List of errors (PlmExceptionType) that occurred during the operation.
- Record limit hit? [true/false] : Indicates whether the query result has hit the mask limit.
- List of objects (PlmObject) found including all the object attribute values listed in the PlmResult of the request.

  Additionally the ID attributes for of the parent and the child record are added. The parent IDs are named PARENT.EDB_ID and PARENT.C_ID, the attributes for child IDs are named CHILD.EDB_ID and CHILD.C_ID.

  This is necessary to prevent name clashes if the parent and child entity are the same (for instance in a item BOM).

  If a count request was made, the response will have one PlmObject with an Integer attribute named COUNT containing the number of objects matching the query, and an Integer attribute called RECORD_LIMIT containing the current record limit of the mask used for the count operation.

Document Management Web Services

A Document is a business object that specifies one or more files that are stored in the Agile e6 File Vault. You can load a document and add one or more files to its Files table. You can also search for a document and its files, just as you would search for an Item.

The Document Management Web Services supports the management of the document-file assignment and the upload/download of files.

Note	To create a document, use the Business Object Web Services.

Streaming support, which is mandatory for the getFileCopy and checkInFile operations to work

Bulk Operations

The following are the bulk operation names of the single request operations for Document Management web services described the concurrent sections:

- getFilesBulk
- getFileCopyBulk

See Processing the Bulk Requests for complete details on how to handle the bulk operations.

getFiles

Service To get a list of files assigned to a specific document.

Usage The request object contains the document handle that identifies the files which are assigned to the document. The response is a list of all files assigned to a document, or an error code.

Request Type GetFilesRequestType

- messageId (String, optional)
- messageName (String, optional)
- plmDocumentFileQuery (PlmDocumentFileQuery):
  - document (PlmObjectRef, required): An object reference to the document (the parent).
  - documentFileMaskName (String, required): The maskname of the document-file relation.
  - selection (List<PlmCondition>, required): Search criteria for object attributes in documentFileMaskName (name/value pairs).
  - plmDfmSite (String, optional): Not supported in this version, ignored if EDB-DFM-ACTIVE=0
  - ignoreRecordLimit (boolean, optional, default=true): Obey record limit? [true/false]
  - excludeVaultValues (boolean, optional, default=false): Should vault name, vault type, vault kind, vaultNode, vaultPath, vaultNetRef and filename in vault of the file not be read and returned? [true/false]
  - attributeNames (List<String>, optional): List of field names to be returned - (optional for the generic result, the attributes which are always returned see in Response section).
  - includeAllLanguages (boolean, optional, default=false): Include all languages? [true/false]
  - includeBinaryValues (boolean, optional, default=false): Include binary values? [true/false]

Response Type GetFilesResponeType

- messageId (String, required)
- messageName (String, required)
- statusCode(ResponseStatusCode, required)
  - SUCCESS, PARTIAL_SUCCESS, WARNING or FAILURE
- exceptions (List<PlmExceptionType>, optional)
  - List of exceptions that occurred during the operation.
- warnings (List<PlmWarningType>, optional)
  - List of warnings that occurred during the operation.
- ticket (String, optional)
  - The Plm ticket can be used instead of the password in subsequent calls. It allows the client to reference the PLM session, even if the HTTP session has been closed, or - in case of a backward flow - if there is no HTTP session.
    The ticket is only valid for the PLM server instance that generated it.
- recordLimitHit (boolean, required)
  - [true | false] Indicates whether the query result has hit the mask limit.
- objects (List<PlmDocumentFileObject>)
  - List of PlmDocumentFileObjects found including all the object attribute values within the generic part as defined in the request

- PlmDocumentFileObject:
  - plmObject(PlmObject)
  - fileOrgName (String), (origin filename)
  - fileOrgNode (String)
  - fileOrgPath (String)
  - fileSize (String)
  - fileModifyDate (Date)
  - fileOs (String) (operating system) (empty_string)
  - fileType (String) (empty_string)
  - fileFormat (String) (empty_string)
  - fileStepCreSystem (String)
  - fileCryptName (String), (empty if excludeVaultValues = true)
  - vaultName (String), (empty if excludeVaultValues = true)
  - vaultType (String), (empty if excludeVaultValues = true)
  - vaultKind (String), (empty if excludeVaultValues = true)
  - vaultNode (String), (empty if excludeVaultValues = true)
  - vaultNetRef (String), (empty if excludeVaultValues = true)
  - vaultPath (String), (empty if excludeVaultValues = true)
- The created objects contain the attributes listed in the request.
  In case one or more of the requested attributes do not exist, PARTIAL_SUCCESS will be returned.
  In case one or more of the requested attributes are not accessible, a WARNING will be returned.
  If no attributes have been requested (attribute list is missing), the whole object including all visible attributes and the ID attributes (EDB_ID and C_ID) will be returned. If none of the requested attributes exists or the list is empty, only the ID attributes (EDB_ID and C_ID) will be returned.

getFileCopy

Service To get a copy of a file that is assigned to a specific document.

Usage This operation does not require any reservation or checking-out of the specified file. The request object contains the document handle, file handles and site ID. The response is the file attributes and link/URL to the files, or an error code.

> ☞ DFM is not supported in this release of Web services. Hence, the getFileCopy operation downloads the file from the specified site even when the file at that site is not current.

Request Type GetFileCopyRequestType

- messageId (String, optional)
- messageName (String, optional)
- plmDocumentFile (PlmDocumentFileObject, required)
  Describes the file to get.This is a PlmDocumentFileObject as returned by the DocumentManagementService.getFiles operation.

Response Type GetFileCopyResponseType

- messageId (String, required)
- messageName (String, required)
- statusCode(ResponseStatusCode, required)
  - SUCCESS, PARTIAL_SUCCESS, WARNING or FAILURE
- exceptions (List<PlmExceptionType>, optional)
  - List of exceptions that occurred during the operation.
- warnings (List<PlmWarningType>, optional)
  - List of warnings that occurred during the operation.
- ticket (String, optional)
  The Plm ticket can be used instead of the password in subsequent calls. It allows the client to reference the PLM session, even if the HTTP session has been closed, or - in case of a backward flow - if there is no HTTP session. The ticket is only valid for the PLM server instance that generated it.
- plmDocumentFile (PlmDocumentFileObject, required)
  The file information.
  This is the same PlmDocumentFileObject that was passed in the request.
- fileData (a javax.activation.DataHandler, optional).
  The file data as a binary MTOM stream with content type "application/octet-stream".
  Use the Streaming API of your JAX-WS implementation to transfer the file. If streaming is not used, the file is transferred in the XML data, which takes considerably longer, leads to huge SOAP messages and might even result in an OutOfMemoryException on the server or the client.

checkInFile

Service To check in a new file.

Usage The request object specifies the document handle, file handles, Site ID, etc of the file that.

A previously checked out files cannot be checked-in with this operation as the request type is completely different.

There is full DFM support for check-in of files via web service. The DFM capabilities apply the same way as if the file would have been checked in via Java Client.

Request Type CheckInFileRequestType

- messageId (String, optional)
- messageName (String, optional)
- plmDocumentFileQuery (PlmDocumentFileQuery):
  - document (PlmObjectRef, required): An object reference to the document (the parent).
  - documentFileMaskName (String, required): The maskname of the document-file relation.
  - selection (List<PlmCondition>, required): Search criteria for object attributes in documentFileMaskName (name/value pairs).
  - plmDfmSite (String, optional): Not supported in this version, ignored if EDB-DFM-ACTIVE=0
  - ignoreRecordLimit (boolean, optional, default=true): Obey record limit? [true/false]
  - excludeVaultValues (boolean, optional, default=false): Should vault name, vault type, vault kind, vaultNode, vaultPath, vaultNetRef and filename in vault of the file not be read and returned? [true/false]
  - attributeNames (List<String>, optional): List of field names to be returned - (optional for the generic result, the attributes which are always returned see in Response section).
  - includeAllLanguages (boolean, optional, default=false): Include all languages? [true/false]
  - includeBinaryValues (boolean, optional, default=false): Include binary values? [true/false]
- documentFileObject (PlmObject, required)
  List of new relationship attributes . The values must be typed values in form of a PlmAttributeChoice.
  The meta data of the relationObject is currently ignored by this operation, so you can pass empty values for these elements.
  Only the list of attribute values is used to fill the new relation record in e6.
  At least following fields and their values must be provided:
  - T_FILE_DAT.STORAGE_AREA (Vault)
  - T_FILE_DAT.ORG_NAME (Original File Name)
  - T_FILE_DAT.ORG_NODE (Node)
  - T_FILE_DAT.ORG_DISCPATH (Original Disk+Path)
- fileData (a javax.activation.DataHandler, optional).
  The file data as a binary MTOM stream with content type "application/octet-stream"

Response Type CheckInFileResponseType

- messageId (String, required)

- messageName (String, required)
- statusCode(ResponseStatusCode, required)
  - SUCCESS, PARTIAL_SUCCESS, WARNING or FAILURE
- exceptions (List<PlmExceptionType>, optional)
  - List of exceptions that occurred during the operation.
- warnings (List<PlmWarningType>, optional)
  - List of warnings that occurred during the operation.
- ticket (String, optional)
  The Plm ticket can be used instead of the password in subsequent calls. It allows the client to reference the PLM session, even if the HTTP session has been closed, or - in case of a backward flow - if there is no HTTP session. The ticket is only valid for the PLM server instance that generated it.
- PlmDocumentFileObject (PlmDocumentFileObject, required):
  - plmObject(PlmObject)
  - fileOrgName (String)
  - fileOrgNode (String)
  - fileOrgPath (String)
  - fileSize (String)
  - fileModifyDate (Date)
  - fileOs (String) (operating system) (empty_string)
  - fileType (String) (empty_string)
  - fileFormat (String) (empty_string)
  - fileStepCreSystem (String)
  - fileCryptName (String), (empty if excludeVaultValues = true)
  - vaultName (String), (empty if excludeVaultValues = true)
  - vaultType (String), (empty if excludeVaultValues = true)
  - vaultKind (String), (empty if excludeVaultValues = true)
  - vaultNode (String), (empty if excludeVaultValues = true)
  - vaultNetRef (String), (empty if excludeVaultValues = true)
  - vaultPath (String), (empty if excludeVaultValues = true)

MetaData Web Services

The Metadata Service enables you to read the definition of Agile e6 classes like entities, entity types and relations.

All operations need one request object as input and return one response object.

- The request contains at least the name of Agile e6 class and a mask name.
- The response contains the metadata of the respective Agile e6 class.
- Bulk operations allow you to execute a whole list of requests with one web service call. The response of a bulk operation contains a list of responses matching the list of requests.

The Agile e6 metadata is read from the Agile e6 application using the mask specified in the request.

It is only possible to access Agile e6 attributes that are visible in this mask, with the exception of the ID fields EDB_ID and C_ID.

Only masks listed in the so called *Web Service White List* of the Agile e6 application can be accessed.

Bulk Operations

The following are the bulk operation names of the single request operations for MetaData web services described the concurrent sections. See Processing the Bulk Requests for complete details on how to handle the bulk operations.

- getEntityBulk
- getEntityTypeBulk
- getEntityRelationBulk

getEntity

Service To get the metadata of a PLM entity.

Usage The data is based on the mask used to access the data. If the request does not pass a specific mask name, the default mask of the entity will be used. The response contains the definition of all visible attributes in the mask. If an attribute has mode specific access, it will be returned regardless of the mode specific access value.

Request Type GetEntityRequestType

- name (String): The entity name.
- mask (String): The mask name to use to read this entity. (optional)
  If empty, the default list of the entity will be used.
  As all other mask names, the mask is checked against the white list for masks. If it is not listed, the access is denied.
- messageId (String): ID to be returned in the response (optional)
- messageName (String) : Name to be returned in the response (optional)

Response Type GetEntityResponseType

- statusCode (ResponseStatusCode) : SUCCESS or FAILURE.
- ticket (String) : A PLM ticket.
  The ticket can be used instead of the password in subsequent calls. It allows the client to reference the PLM session, even if the HTTP session has been closed, or - in case of a backward flow - if there is no HTTP session.
  The ticket is only valid for the PLM server instance that generated it.
- messageId (String): copied from the request, or generated.
- messageName (String): copied from the request, or the operation name.
- warnings: List of warnings (PlmWarningType) that occurred during the operation.
- exceptions: List of exceptions (PlmExceptionType) that occurred during the operation.
- entity (PlmEntity): contains the entity definition (all information is returned with type String, unless noted otherwise): :
  - Entity ID.
  - Entity name.
  - Mask name.
  - Default form.
  - Default list.
  - Table name.
  - Join table name.
  - Entity title (in the current session language).
  - Mask title (in the current session language).
  - Record limit.
  - Number of significant fields.
  - List of attribute definitions (PlmMetaAttribute) containing:
    - Attribute name
    - Attribute type.
    - Attribute title (in the current session language),
    - Attribute description (in the current session language).
    - Attribute format.
    - Attribute default value.

    - Attribute Checkstring
    - Attribute access (visible/invisible/mandatory/read-only/mode specific):
        - Query mode specific attribute access (if applicable).
        - Update mode specific attribute access (if applicable).
        - Insert mode specific attribute access (if applicable).
    - Data size of the attribute in this mask.
    - Visible field width in this mask. (useful if a UI is generated from this meta data)
    - Visible field height in this mask. (useful if a UI is generated from this meta data)
    - Attribute visible? Tells you if the attribute is visible in this mask.
- List of relation meta data (PlmMetaRelation) defined for this entity:
    - The name of the parent entity.
    - The name of the child entity.
    - The relation type.
    - The name of the view.
    - The name of the relation.
    - The internal ID of the relation.
    - The table name.
    - The title of the relation (in the current session language).
    - The default list.
    - The default form (empty for most relations).
    - The mask name used to read the meta data.

  This data can later be used to call MetadataService.getEntityRelation to get all attribute meta data for one specific relation.
- List of type meta data (PlmMetaType) defined for this entity:
    - The name of the master entity.
    - The type name.
    - The table name.
    - The title of the type
    - The default list.
    - The default form.
    - The mask name used to read the meta data.

  This data can later be used to call MetadataService.getEntityType to get all attribute meta data for one specific type.

getEntityType

Service To get the metadata of a PLM entity type.

Usage The data is based on the mask used to access the data. If the request does not pass a specific mask name, the default mask of the entity type will be used. The response contains the definition of all visible attributes in the mask. If an attribute has mode specific access, it will be returned regardless of the mode specific access value.

If the mask contains visible multi-language attributes, all generated invisible multi-language attribute siblings are returned.

Request Type GetEntityTypeRequestType

- name (String): the name of the master entity for this type
- type (String): the name of the type
- mask (String): The mask name to use to read this entity. (optional)
  If empty, the default list of the entity will be used.
  As all other mask names, the mask is checked against the white list for masks. If it is not listed, the access is denied.
- messageId (String): ID to be returned in the response (optional)
- messageName (String) : Name to be returned in the response (optional)

Response Type GetEntityTypeResponseType

- statusCode (ResponseStatusCode) : SUCCESS or FAILURE.
- ticket (String) : A PLM ticket.
  The ticket can be used instead of the password in subsequent calls. It allows the client to reference the PLM session, even if the HTTP session has been closed, or - in case of a backward flow - if there is no HTTP session.
  The ticket is only valid for the PLM server instance that generated it.
- messageId (String): copied from the request, or generated.
- messageName (String): copied from the request, or the operation name.
- warnings: List of warnings (PlmWarningType) that occurred during the operation.
- exceptions: List of exceptions (PlmExceptionType) that occurred during the operation.
- An entity type definition of type PlmEntityType (all information is returned with type String, unless noted otherwise):
  - Entity name
  - Type name
  - Mask name
  - Table name
  - Join table name
  - Entity type title (in the current session language).
  - Mask title (in the current session language).
  - Mask limit.
  - Number of significant fields
  - Master: the reference to the master entity as a PlmClassRef
  - List of attribute definitions containing:
    - Attribute type.
    - Attribute title (in the current session language),
    - Attribute description (in the current session language).
    - Attribute format.

  - Attribute default value.
  - Attribute access (visible/invisible/mandatory/read-only/mode specific):
    - Query mode specific attribute access (if applicable).
    - Update mode specific attribute access (if applicable).
    - Insert mode specific attribute access (if applicable).
  - Data size of the attribute in this mask.
  - Visible field width in this mask. (useful if a UI is generated from this meta data)
  - Visible field height in this mask. (useful if a UI is generated from this meta data)
  - Attribute visible? Tells you if the attribute is visible in this mask.
- List of relation meta data (PlmMetaRelation) defined for this entity:
  - The name of the parent entity.
  - The name of the child entity.
  - The relation type.
  - The name of the view.
  - The name of the relation.
  - The internal ID of the relation.
  - The table name.
  - The title of the relation (in the current session language).
  - The default list.
  - The default form.

  This data can later be used to call MetadataService.getEntityRelation to get all attribute meta data for one specific relation.

getEntityRelation

Service To get the metadata of a PLM constraint, refine or aggregate relation.

Usage The meta relation object can either be taken from the response of a call to getEntity/getEntity type, or it can be created using hard coded default values. The request attributes – parent, child, type and view are needed to identify the relation data.

Request Type GetEntityRelationRequestType

- metaRelation (PlmMetaRelation): Describes the relation to read and must contain at least the following information:
  - parent: The name of the parent entity. (e.g. "EDB-ARTICLE")
  - child: The name of the child entity (e.g. "EDB-DOCUMENT")
  - type: The relation type. (e.g. PlmRelationTypeEnum.REFINE)
  - view: The name of the view. (e.g. "STR").
    The meta relation object can either be taken from the response of a call to getEntity/getEntity type, or it can be created using hard coded default values. The four attributes listed above are needed to identify the relation data, the rest of the attributes in PlmMetaRelation are not relevant.
- messageId (String): ID to be returned in the response (optional)
- messageName (String) : Name to be returned in the response (optional)

Response Type GetEntityRelationResponseType

- statusCode (ResponseStatusCode) : SUCCESS or FAILURE.
- ticket (String) : A PLM ticket.
  The ticket can be used instead of the password in subsequent calls. It allows the client to reference the PLM session, even if the HTTP session has been closed, or - in case of a backward flow - if there is no HTTP session.
- The ticket is only valid for the PLM server instance that generated it.
- messageId (String): copied from the request, or generated.
- messageName (String): copied from the request, or the operation name.
- warnings: List of warnings (PlmWarningType) that occurred during the operation.
- exceptions: List of exceptions (PlmExceptionType) that occurred during the operation.
- plmRelation (PlmRelation) : The relation definition containing:
  - Relation name.
  - PlmMetaRelation: Detailed meta information of the relation.
  - Mask name.
  - Table name.
  - Entity type title (in the current session language).
  - Mask title (in the current session language).
  - Mask limit.
  - Number of significant fields.
  - List of attribute definitions containing:
    - Attribute type.
    - Attribute title (in the current session language),
    - Attribute description (in the current session language).
    - Attribute format.
    - Attribute default value.
    - Attribute access (visible/invisible/mandatory/read-only/mode specific):
      - Query mode specific attribute access (if applicable).

    - Update mode specific attribute access (if applicable).
    - Insert mode specific attribute access (if applicable).
- Data size of the attribute in this mask.
- Visible field width in this mask. (useful if a UI is generated from this meta data)
- Visible field height in this mask. (useful if a UI is generated from this meta data)
- Attribute visible? Tells you if the attribute is visible in this mask.

Configuration Web Services

The Configuration Web Service enables you to retrieve PLM objects from the Agile e6 application. It retrieves configuration data of a PLM object, such as *Default, User Context, etc* specified by its name.

The requests include the value(s) for specifying the requested object. Responses include the requested objects.

Bulk operations require a list of requests to execute. These return with a list of responses.

Bulk Operations

The following are the bulk operation names of the single request operations for Configuration web services described the concurrent sections. See Processing the Bulk Requests for complete details on how to handle the bulk operations.

- getDefaultBulk

getUserContext

Service To get all the information of the current user context of a PLM session.

Usage The request object carries current PLM user's details and only an optional message ID and name. The response object delivers the user attributes, all group assignments attributes, current view (Released, Global etc.), current context (DSG, ENG), current project assignment (Project ID) and current Org assignment (Org ID).

Request Type GetUserContextRequestType

- messageId (String): ID to be returned in the response (optional)
- messageName (String) : Name to be returned in the response (optional)

Response Type GetUserContextResponseType

- statusCode (ResponseStatusCode) : SUCCESS or FAILURE.
  - SUCCESS: all objects from user context are returned, if they are available. There could be some warnings in the message block, if some objects(e.g. currentJob, currentRole) are not available. It could make sense for setUserContext, but is not necessary to check for getUserContext.
  - FAILURE: there was connection problem.
- ticket (String) : A PLM ticket.
  The ticket can be used instead of the password in subsequent calls. It allows the client to reference the PLM session, even if the HTTP session has been closed, or - in case of a backward flow - if there is no HTTP session.
  The ticket is only valid for the PLM server instance that generated it.
- messageId (String): copied from the request, or generated.
- messageName (String): copied from the request, or the operation name.
- warnings: List of warnings (PlmWarningType) that occurred during the operation.
- exceptions: List of exceptions (PlmExceptionType) that occurred during the operation.
- plmUserContext (PlmUserContext):
  - plmUserInfo (PlmUserContextUserInfo):
    - userName
    - userID
    - group
    - groupID
    - userLanguage
    - userLocale
  - plmViewInfo (PlmUserContextView):
    - preliminaryFlag (Boolean.TRUE, Boolean.False, null)
    - referenceDate (yyyy-MM-dd HH:mm:ss, @NOW, empty_string)
    - versionViewTitle (Current, Production, Global, Development_extended)
    - versionView (PlmVersionViewEnum)
  - plmChangeManagementInfo (PlmUserContextChg):
    - currentChgFlag (Boolean.TRUE, Boolean.False,null)
    - currentWorkOrder (EDB_ID, empty_string)
    - currentWorkSet (EDB_ID, empty_string)
  - plmMoaMpaInfo (PlmUserContextMoaMpa) :
    - moaMpaConfFlag(PlmUserContextMOAEnum, null)

    - currentJob (empty_string)
    - currentRole (empty_string)
    - currentProjectOrOrganisation (empty_string)
    - currentProjectOrOrganisationCid (empty_string)
- plmDFMInfo (PlmUserContextDFM):
    - dfmConfigFlag (Boolean.TRUE, Boolean.False,null)
    - site (Reserved for future use)

setUserContext

Service To set the current user context to be used in the next operations for the current PLM session.

Usage This operation is used to set current job in MOA/MPA environment. The request object carries the user context details, such as the new view, new assignment, new Org assignment and new Project assignment. The response object delivers the user attributes, group assignments, new current view, new current context, new current project assignment, new current org assignment.

Request Type SetUserContextRequestType

- messageId (String): ID to be returned in the response
- messageName (String) : Name to be returned in the response
- plmUserContext (PlmUserContext):
  - plmUserInfo (PlmUserContextUserInfo):
    - userName (not changeable by current user)
    - userID (not changeable by current user)
    - group (Reserved for future use : currently there is no API to implement it.)
    - groupID (Reserved for future use : currently there is no API to implement it.)
    - userLanguage (e.g. ENG, GER, FRA)
    - userLocale (always changed with user language)
  - plmViewInfo (PlmUserContextView):
    - preliminaryFlag (Boolean.TRUE, Boolean.FALSE)
    - referenceDate (in the form of yyyy-MM-dd HH:mm:ss)
    - versionViewTitle (ignored)
    - versionView (PlmVersionViewEnum)
  - plmChangeManagementInfo (PlmUserContextChg):
    - currentChgFlag (not changeable by current user)
    - currentWorkOrder (EDB-ID of work order, no check of currentChgFlag)
    - currentWorkSet (EDB-ID of work set, no check of currentChgFlag)
  - plmMoaMpaInfo (PlmUserContextMoaMpa) :
    - moaMpaConfFlag (not changeable by current user)
    - currentJob EDB_ID of the job, access will be checked by e6 server
    - currentProjectOrOrganisation - ignored in the request, filled by the response
    - currentProjectOrOrganisationCid - ignored in the request, filled by the response
    - currentRole - ignored in the request, filled by the response
  - plmDFMInfo (PlmUserContextDFM):
    - dFMConfigFlag (not changeable by current user)
    - site (Reserved for future use)

getDefault

Service To get the contents of a PLM Default value.

Usage The request object carries only the default name. In case the value cannot be parsed (e.g. letter instead of number as integer), this value is taken as 'null'.

Request Type GetDefaultRequestType

- defaultName (String): Name of the DataView default.
- messageId (String): ID to be returned in the response (optional)
- messageName (String) : Name to be returned in the response (optional)

Response Type GetDefaultResponseType

- Status Code
  - Successful
  - Partial successful: Typed default value is NULL
  - Failure: Default "Non-existent" - PLM Object is NULL~,
- PLM Default as complex data type (String, Integer, Float, Boolean)