CentraSite

Importing Objects Using the API

Version 9.5 SP1

November 2013

This document applies to CentraSite Version 9.5 SP1.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: IINM-OIINMAG-IMPORT-API-95SP1-20140410

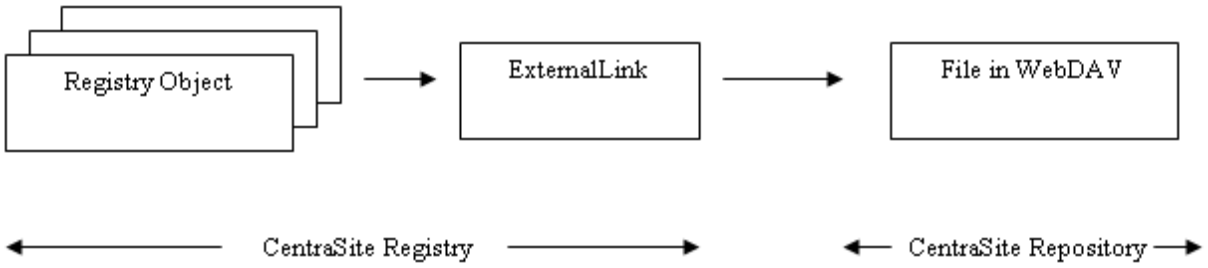
Table of Contents

Preface

CentraSite has a set of importers for importing various kinds of objects. They are used by the Import function of CentraSite Control, but they can also be invoked from Java programs, through the SOAP API, from batch commands (e.g. shell scripts) or directly from the command-line prompt. Import APIs are available for importing:

- Web services
- XML schemas
- XML services
- REST services
- BPEL process files
- XPDL files

All the importers have a few characteristics in common. Each import function is called with an appropriate XML file (for example, the web service importer expects a WSDL file as input). The XML file can be located in the file system, or, for some importers, it can alternatively be specified by a URL (HTTP). The file is incorporated into the CentraSite repository, and there will be a registry object (an ExternalLink) that will point to it. Each importer typically creates one or more JAXR-based registry objects. The following picture illustrates the relationship:



This document discusses the following topics:

CentraSite Importers

Invoking an Importer from a Java Program	Describes how an importer can be invoked using the CentraSite Java API.
Invoking an Importer from a Java Class	Describes how the importers can be invoked from a Java class.
Invoking an Importer from the Command Line	Describes how the importers can be invoked from the command line or a batch script.
Invoking an Importer from the SOAP API	Describes how to access the importers using Web services.

Writing Your Own Importer

•	**Writing Your Own Importer**	Describes how to write an importer and integrate it into the CentraSite environment.

1 Invoking an Importer from a Java Program

The classes used by the importers are contained in the file *CentraSiteUtils.jar*. The `CLASSPATH` variable must refer to the JAR files that are used by CentraSite. It is convenient to include all *JAR* files contained in the *redist* folder of the CentraSite installation.

After using an instance of the `JAXRAccessor` class to establish a valid connection (with user and password) to CentraSite, one or more imports are possible. Finally, close the session by calling the `close()` method of `JAXRAccessor`.

The following topics are discussed below:

Importing a Web Service	**Updating a Registered Web Service**	**Attaching a WSDL to a Registered Web Service**	**Removing a Registered Web Service**	**Finding a Registered Web Service**
Importing a Schema			**Removing a Schema**	
Importing an XML Service			**Removing an XML Service**	
Importing a BPEL Process File			**Removing a Registered BPEL Object**	
Importing an XPDL File				

Importing a Web Service

The basic input when importing a web service is a WSDL file describing one or more services. There are also some setter methods for additional parameters. The organization (per object or per name) must be set; other parameters are optional.

The following example demonstrates how to register a web service to CentraSite:

```
import com.centrasite.jaxr.JAXRAccessor;
import com.centrasite.jaxr.webservice.WebServiceRegistrator;

String dbURL = "http://localhost:53307/CentraSite/CentraSite";
String organizationName = "MyOrganization";
String wsdlFile = "c:/temp/MyService.wsdl";
String user = ...;
String password = ...;

JAXRAccessor jaxr = new JAXRAccessor(dbURL, user, password);

try
{
   WebServiceRegistrator wsr = new WebServiceRegistrator(wsdlFile, jaxr);
   wsr.setOrganization(organizationName);
   wsr.register();
```

```
}
catch (Exception e)
{
   // handle error
}
finally
{
   jaxr.close();
}
```

The `WebServiceRegistrator` class provides the setter methods listed below. The setter methods must be called before calling the `register()` method.

setDescription

Sets a description text that will appear with the imported `Service` object.

Syntax

```
public void setDescription(String description)
```

Parameters

String description
The description text.

Usage Notes

Optional.

setOrganization

Sets the name of the organization under which the service should be registered.

Syntax

```
public void setOrganization(String organizationName)
```

Parameters

String organizationName
The name of the organization.

setOrganization

Sets the organization object under which the service should be registered.

Syntax

```
public void setOrganization(Organization organizationObject)
```

Parameters

Organization organizationObject
The organization object under which the service should be registered.

setPackageName

Sets the name of the registry package of which the service should be a member.

Syntax

```
public void setPackageName(String packageName)
```

Parameters

String packageName
The name of the registry package of which the service should be a member.

Usage Notes

Optional.

setPackageObject

Sets the registry package object of which the service should be a member.

Syntax

```
public void setPackageObject(RegistryPackage packageObject)
```

Parameters

RegistryPackage packageObject
The registry package object of which the service should be a member.

Usage Notes

Optional.

setUsedObject

Sets a registry object that will point to the imported service with a "Uses" association; in other words, the imported service will be used by the specified object.

Syntax

```
public void setUsedObject(RegistryObject usedObject)
```

Parameters

RegistryObject usedObject
The registry object that will point to the imported service.

Usage Notes

Optional.

setUsesObject

Sets a registry object that will be pointed to by the imported service with a "Uses" association; in other words, the imported service will use the specified object.

Syntax

```
public void setUsesObject(RegistryObject usesObject)
```

Parameters

RegistryObject usesObject
The registry object that will be pointed to by the imported service.

Usage Notes

Optional.

setWebDAVFolder

Changes the path of the folder where the WSDL file is stored in the CentraSite repository.

Syntax

```
public void setWebDAVFolder(String folderPath)
```

Parameters

String folderPath
The path of the folder where the WSDL file should be stored.

Default: */projects/WSDL*.

Usage Notes

Optional.

Updating a Registered Web Service

To update a registered web service (for example, when the WSDL file has been modified or if you want to change the registry package that contains the web service), call the `register()` method with the modified WSDL file – see the **example above** for details.

Note: The key associated with a registered service comprises its organization, its name and the WSDL file's targetNamespace. If you modify one or more of these, the service is not updated; rather, a new registry entry is created.

Attaching a WSDL to a Registered Web Service

It is possible to attach a WSDL file to an existing web service. The code for attaching a WSDL file is similar to the **code for importing a WSDL file**. An example code snippet follows:

```
WebServiceRegistrator wr = new WebServiceRegistrator("attach.wsdl", jaxr);
wr.setAttachServiceID("uddi...");
wr.register();
```

where `"uddi..."` denotes the UDDI key of the service to which the WSDL file should be attached.

Note the following:

- The service can be a manually-created service.
- If the service has already been registered with a WSDL, then calling `setAttachServiceID` updates the registered information.
- If there are any services within the WSDL, they are registered as usual.
- If the service has a ServiceBinding that is not in the WSDL, it is retained.
- If the service has a ServiceBinding that is in the WSDL, it is updated. Operations are never deleted.

The `WebServiceRegistrator` class provides the setter method listed below (in addition to the **setter methods described under Importing a Web Service**). The setter method must be called before calling the `register()` method.

setAttachServiceID

Sets a Service ID (`getKey().getId()`) for a WSDL-service attachment.

Syntax

```
public final void setAttachServiceID(java.lang.String attachServiceID)
```

Parameters

java.lang.String attachServiceID)
The ID of the service, in the form `"uddi:..."`.

Removing a Registered Web Service

Use the functions described below to remove a registered web service and all its resources, including all associated registry objects and the WSDL file in the CentraSite repository.

You must use an instance of the `JAXRAccessor` class to establish a CentraSite connection before calling the `removeService` or `removeServiceByID` function. Finally, close the session by calling the `close()` method of `JAXRAccessor`.

Also before calling the `removeService` or `removeServiceByID` function, you may call setter functions, which are also described below.

The following example demonstrates how to remove a web service from CentraSiteCentraSite:

```
import com.centrasite.jaxr.JAXRAccessor;
import com.centrasite.jaxr.webservice.WebServiceAdministrator;

String dbURL = "http://localhost:53307/CentraSite/CentraSite";
String wsdlFile = "c:/temp/MyService.wsdl";
String user = ...;
String password = ...;

JAXRAccessor jaxr = new JAXRAccessor(dbURL, user, password);

try
{
   WebServiceAdministrator wsa = new WebServiceAdministrator(jaxr);
   int removeCount = wsa.removeServices(wsdlFile);
}
catch (Exception e)
{
```

```
   // handle error
}
finally
{
   jaxr.close();
}
```

The `WebServiceAdministrator` class provides the methods shown below:

removeService

Removes the service specified by the unique name and namespace.

Syntax

```
public boolean removeService(String serviceName, String namespace)
```

Parameters

String serviceName
The name of the service to be removed.

String namespace
The namespace of the service to be removed.

Return Codes

Value	Meaning
true	The specified service was successfully found and removed;
false	otherwise.

removeServices

Removes from CentraSite all services that are indicated by the specified WSDL file.

Syntax

```
public int removeServices(String wsdlFilename)
```

Parameters

String wsdlFilename
The name of the WSDL file that indicates the services to be removed.

Return Codes

Value	Meaning
int	The number of services that were removed.

removeServiceByID

Removes the service with the specified ID.

Syntax

```
public boolean removeServiceByID(String serviceID)
```

Parameters

String serviceID

The ID of the service to be removed. The ID is the `getKey().getID()` value of the associated service object.

Return Codes

Value	Meaning
true	The specified service was successfully found and removed;
false	otherwise.

setDeleteTargetAssocs

Specifies whether or not to delete associations to the service object where the service object is the target of the association. If the parameter is "false" and the service object is the target of one or more associations, the removal of the service object is inhibited.

Syntax

```
public void setDeleteTargetAssocs(boolean deleteTargetAssocs)
```

Parameters

boolean deleteTargetAssocs

true: Delete associations to the service object where the service object is the target of the association.

Default: true.

setRemoveReferencedImports

Specifies whether or not to remove from the repository referenced imported WSDL and schema files, together with their controlling ExternalLinks.

Syntax

```
public void setRemoveReferencedImports(boolean removeReferencedImports)
```

Parameters

boolean removeReferencedImports
true: Remove the referenced imported WSDL and schema files.

setRemoveWsdl

Specifies whether or not the WSDL file together with its controlling ExternalLink should also be removed from the repository.

Syntax

```
public void setRemoveWsdl(boolean removeWsdl)
```

Parameters

boolean removeWsdl
true: Remove the WSDL file together with its controlling ExternalLink from the repository.

Default: true.

Finding a Registered Web Service

The Web Service API provides functions with which you can find a registered web service. The following example demonstrates this:

```
import com.centrasite.jaxr.JAXRAccessor;
import com.centrasite.jaxr.webservice.WebServiceAdministrator;

String dbURL = "http://localhost:53307/CentraSite/CentraSite";
String wsdlFile = "c:/temp/MyService.wsdl";
String user = ...;
String password = ...;

JAXRAccessor jaxr = new JAXRAccessor(dbURL, user, password);

try
{
   WebServiceAdministrator wsa = new WebServiceAdministrator(jaxr);
   Collection services = wsa.findWebServices(wsdlFile);  // Service object coll.
   . . .
}
catch (Exception e)
{
   // handle error
}
finally
{
   jaxr.close();
}
```

The `WebServiceAdministrator` class provides the methods shown below:

findWebServiceByNamespace

Finds the registered web service on the basis of its name and namespace.

Syntax

```
public Service findWebServiceByNamespace(String serviceName, String namespace)
```

Parameters

String serviceName
The name of the registered web service.

String namespace
The namespace of the registered web service.

findWebServices

Finds a collection of web service objects that are registered with the specified organization in the specified WSDL file.

Syntax

```
public Collection findWebServices(String organizationName, String wsdlFile)
```

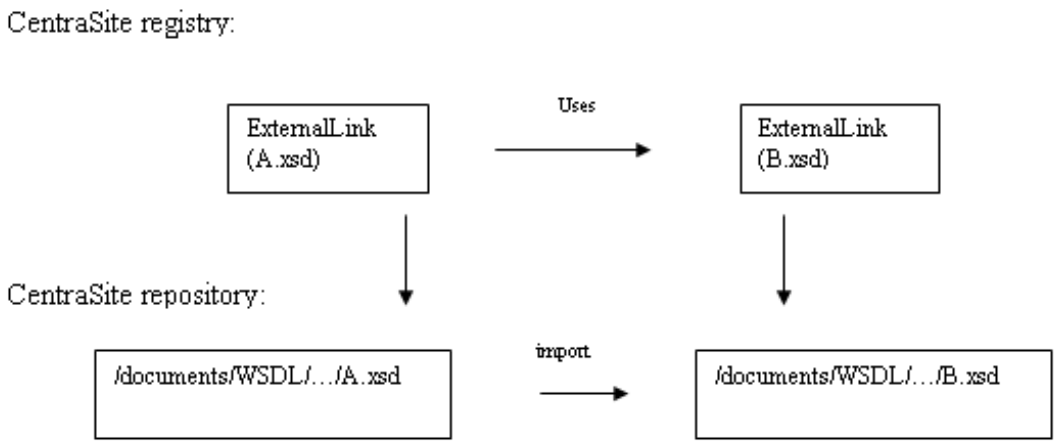
Parameters

String organizationName
The name of the organization.

String wsdlFile
The filepath to the WSDL file.

Importing a Schema

The schema importer works closely together with the **web service importer**. Since a WSDL file of a web service may import or include schema files, CentraSite enables you to design and store a schema before referencing WSDL files. Similarly, if a schema file is imported by more than one WSDL file, it could be beneficial to register the schema first, before registering the WSDL files. When a schema is imported, the file is copied into the CentraSite repository and an ExternalLink that controls the resource is created. If a schema imports (includes) further schemas, then the referenced schemas are also imported. The referenced relations are established by means of a "Uses" association in the registry. The following picture illustrates the relationship:



The following example demonstrates how to import a schema file into CentraSite:

```
import com.centrasite.jaxr.JAXRAccessor;
import com.centrasite.jaxr.schema.SchemaImporter;

String dbURL = "http://localhost:53307/CentraSite/CentraSite";
String xsdFile = "c:/temp/MySchema.xsd";
String user = ...;
String password = ...;

JAXRAccessor jaxr = new JAXRAccessor(dbURL, user, password);

try
{
   SchemaImporter si = new SchemaImporter(xsdFile, jaxr);
   si.add();
}
catch (Exception e)
{
   // handle error
}
finally
{
   jaxr.close();
}
```

Removing a Schema

You can remove a schema from CentraSite. This function deletes the XML Schema object, the ExternalLink and the resource in the repository.

Note: If a schema is referenced by another object, for example if it is referenced by the ExternalLink of a WSDL object, then it cannot be deleted.

The following example demonstrates how to remove a schema from CentraSite:

```
import com.centrasite.jaxr.JAXRAccessor;
import com.centrasite.jaxr.schema.SchemaAdministrator;
import javax.xml.registry.infomodel.ExternalLink;

String dbURL = "http://localhost:53307/CentraSite/CentraSite";
String user = ...;
String password = ...;
ExternalLink schemaExtLink = ...;

JAXRAccessor jaxr = new JAXRAccessor(dbURL, user, password);

try
```

```
{
   SchemaAdministrator sa = new SchemaAdministrator(jaxr);
   sa.remove(schemaExtLink);
}
catch (Exception e)
{
   // handle error
}
finally
{
   jaxr.close();
}
```

The `SchemaAdministrator` class provides the following methods for removing a schema:

remove

Removes the schema at the specified location in the CentraSite repository, which may be relative or absolute.

Syntax

```
public boolean remove(String schemaLocation)
```

Parameters

String schemaLocation
The repository location, which may be relative or absolute.

Return Codes

Value	Meaning
true	The specified schema was found and removed.

remove

Removes the schema specified by its XML schema object.

Syntax

```
public boolean remove(RegistryObject schemaObject)
```

Parameters

RegistryObject schemaObject
The XML schema object or the external link of the schema.

Return Codes

Value	Meaning
true	The specified schema was found and removed.

removeCascading

Removes the specified schema and also all schemas that are related to it by import and/or include. The initial schema is specified by its location in the CentraSite repository, which may be relative or absolute.

Syntax

```
public boolean removeCascading(String schemaLocation)
```

Parameters

String schemaLocation
The repository location, which may be relative or absolute.

Return Codes

Value	Meaning
true	The specified schema and all related schemas were found and removed.

removeCascading

Removes the specified schema and also all schemas that are related to it by import and/or include. The initial schema is specified by its XML schema object or by its external link.

Syntax

```
public boolean removeCascading(RegistryObject schemaObject)
```

Parameters

RegistryObject schemaObject
The XML schema object or the external link of the schema.

Return Codes

Value	Meaning
true	The specified schema and all related schemas were found and removed.

Importing an XML Service

The XML service importer utilizes an XML Schema Documentation (XSD) as the basic input. The importer internally works with the **schema importer** for importing the schema file.

The following example demonstrates how to import an XML Service to CentraSite:

```
import javax.xml.registry.Connection;
import javax.xml.registry.JAXRException;
import javax.xml.registry.infomodel.Organization;
import com.centrasite.jaxr.xmlservice.HttpMethods;
import com.centrasite.jaxr.xmlservice.XMLService;
import com.centrasite.jaxr.xmlservice.XMLServiceInfo;
import com.centrasite.jaxr.xmlservice.XMLServiceManager;

String dbUrl = "http://localhost:53307/CentraSite/CentraSite";
String user = "...";
String password = "...";
String xsdFile = "C:/temp/myschema.xsd";

// create the required objects
Connection connection = createCentrasiteConnection(dbUrl, user, password);
Organization submittingOrganization = getSubmittingOrganization();
Organization providingOrganization = getProvidingOrganization();

// http methods to be supported
Collection<HttpMethods> httpMethods = new HashSet<HttpMethods>();
```

```
httpMethods.add(HttpMethods.GET);
httpMethods.add(HttpMethods.POST);

// configuration
XMLServiceInfo xmlServiceInfo = new XMLServiceInfo();
xmlServiceInfo.setName("service name");
xmlServiceInfo.setDescription("service description");
xmlServiceInfo.setVersion("1.1");
xmlServiceInfo.setSubmittingOrganization(submittingOrganization);
xmlServiceInfo.setProvidingOrganization(providingOrganization);
xmlServiceInfo.setSchemaUrl(xsdFile);
xmlServiceInfo.setEndpoint("http://endpointurl");
xmlServiceInfo.setHttpMethods(httpMethods);

// create XMLService
XMLServiceManager xmlServiceManager = new XMLServiceManager(connection);
XMLService xmlService = xmlServiceManager.createXMLService(xmlServiceInfo);
```

There are also some setter methods for additional parameters. The name, organization, native service URL, HTTP method must be set; other parameters are optional.

The `XMLServiceManager` class provides the setter methods listed below.

setName

Sets a name text that will appear with the imported `Service` object.

Syntax

```
xmlServiceInfo.setName(String serviceName)
```

Parameters

String serviceName
Name for the service.

setDescription

Sets a description text that will appear with the imported `Service` object.

Syntax

```
xmlServiceInfo.setDescription(String serviceDescription)
```

Parameters

String serviceDescription
The description text.

Usage Notes

Optional.

setVersion

Sets a version number that will appear with the imported `Service` object.

Syntax

```
xmlServiceInfo.setVersion(String version)
```

Parameters

String version
An initial version identifier of the service.

Usage Notes

Optional.

setSubmittingOrganization

Sets the name of the organization under which the service should be registered for the purposes of governance within CentraSite.

Syntax

```
xmlServiceInfo.setSubmittingOrganization(Organization submittingOrganization)
```

Parameters

Organization submittingOrganization
The name of the submitting organization.

setProvidingOrganization

Sets the name of the organization under which the service should be registered from a business perspective.

Syntax

```
xmlServiceInfo.setProvidingOrganization(Organization providingOrganization)
```

Parameters

Organization providingOrganization
The name of the providing organization.

setSchemaURL

Sets the URL of a native XML service to route the requests.

Syntax

```
xmlServiceInfo.setSchemaUrl(String xsdFileUrl)
```

Parameters

String xsdFileUrl
URL to the schema file of the native XML service.

setEndpoint

Sets the schema endpoint of the native XML service to which the requests should be routed.

Syntax

```
xmlServiceInfo.setEndpoint(String endpoint)
```

Parameters

String endpoint
The native XML service's endpoint.

setHttpMethods

Sets a protocol over which the native service should accept requests.

Syntax

```
xmlServiceInfo.setHttpMethods(Collection<HttpMethods> httpmethods)
```

Parameters

Collection<HttpMethods> httpmethods
The protocol to route the requests

Removing an XML Service

You can remove an XML service from CentraSite. This function deletes the XML service and the associated schemas in the repository

Note: If an XML service is referenced by another object, then it cannot be deleted.

The following example demonstrates how to remove an XML service from CentraSite:

```
String xmlServiceKey = "uddi:...";
XMLServiceManager xmlServiceManager = new XMLServiceManager(connection);
xmlServiceManager.deleteXMLService(xmlServiceKey);
```

The `XMLServiceManager` class provides the following methods for removing an XML service:

String xmlServiceKey

Sets a UDDI key for the native XML service.

Syntax

```
XMLServiceManager.deleteXMLService(String xmlServiceKey)
```

Parameters

String xmlServiceKey
UDDI key of the native XML service.

Importing a REST Service

The REST service importer utilizes an XML Schema Documentation (XSD) as the basic input. The importer internally works with the **schema importer** for importing the schema file.

The following example demonstrates how to import an XML Service to CentraSite:

```
import javax.xml.registry.Connection;
import javax.xml.registry.JAXRException;
import javax.xml.registry.infomodel.Organization;
import com.centrasite.jaxr.xmlservice.HttpMethods;
import com.centrasite.jaxr.xmlservice.XMLService;
import com.centrasite.jaxr.xmlservice.XMLServiceInfo;
import com.centrasite.jaxr.xmlservice.XMLServiceManager;

String dbUrl = "http://localhost:53307/CentraSite/CentraSite";
String user = "...";
String password = "...";
String xsdFile = "C:/temp/myschema.xsd";

// create the required objects
Connection connection = createCentrasiteConnection(dbUrl, user, password);
Organization submittingOrganization = getSubmittingOrganization();
Organization providingOrganization = getProvidingOrganization();

// http methods to be supported
Collection<HttpMethods> httpMethods = new HashSet<HttpMethods>();
httpMethods.add(HttpMethods.GET);
httpMethods.add(HttpMethods.POST);

// configuration
XMLServiceInfo xmlServiceInfo = new XMLServiceInfo();
xmlServiceInfo.setName("service name");
```

```
xmlServiceInfo.setDescription("service description");
xmlServiceInfo.setVersion("1.1");
xmlServiceInfo.setSubmittingOrganization(submittingOrganization);
xmlServiceInfo.setProvidingOrganization(providingOrganization);
xmlServiceInfo.setSchemaUrl(xsdFile);
xmlServiceInfo.setEndpoint("http://endpointurl");
xmlServiceInfo.setHttpMethods(httpMethods);

// create XMLService
XMLServiceManager xmlServiceManager = new XMLServiceManager(connection);
XMLService xmlService = xmlServiceManager.createXMLService(xmlServiceInfo);
```

There are also some setter methods for additional parameters. The name, organization, native service URL, HTTP method must be set; other parameters are optional.

The `XMLServiceManager` class provides the setter methods listed below.

setName

Sets a name text that will appear with the imported `Service` object.

Syntax

```
xmlServiceInfo.setName(String serviceName)
```

Parameters

String serviceName
Name for the service.

setDescription

Sets a description text that will appear with the imported `Service` object.

Syntax

```
xmlServiceInfo.setDescription(String serviceDescription)
```

Parameters

String serviceDescription
The description text.

Usage Notes

Optional.

setVersion

Sets a version number that will appear with the imported `Service` object.

Syntax

```
xmlServiceInfo.setVersion(String version)
```

Parameters

String version
An initial version identifier of the service.

Usage Notes

Optional.

setSubmittingOrganization

Sets the name of the organization under which the service should be registered for the purposes of governance within CentraSite.

Syntax

```
xmlServiceInfo.setSubmittingOrganization(Organization submittingOrganization)
```

Parameters

Organization submittingOrganization
The name of the submitting organization.

setProvidingOrganization

Sets the name of the organization under which the service should be registered from a business perspective.

Syntax

```
xmlServiceInfo.setProvidingOrganization(Organization providingOrganization)
```

Parameters

Organization providingOrganization
The name of the providing organization.

setSchemaURL

Sets the URL of a native REST service to route the requests.

Syntax

```
xmlServiceInfo.setSchemaUrl(String xsdFileUrl)
```

Parameters

String xsdFileUrl
URL to the schema file of the native REST service.

setEndpoint

Sets the schema endpoint of the native REST service to which the requests should be routed.

Syntax

```
xmlServiceInfo.setEndpoint(String endpoint)
```

Parameters

String endpoint
The native REST service's endpoint.

setHttpMethods

Sets a protocol over which the native REST service should accept requests.

Syntax

```
xmlServiceInfo.setHttpMethods(Collection<HttpMethods> httpmethods)
```

Parameters

Collection<HttpMethods> httpmethods
The protocol to route the requests

Removing a REST Service

You can remove a REST service from CentraSite. This function deletes the REST service and the associated schemas in the repository

Note: If the REST service is referenced by another object, then it cannot be deleted.

The following example demonstrates how to remove a REST service from CentraSite:

```
String xmlServiceKey = "uddi:...";
XMLServiceManager xmlServiceManager = new XMLServiceManager(connection);
xmlServiceManager.deleteXMLService(xmlServiceKey);
```

The `XMLServiceManager` class provides the following methods for removing an REST service:

String xmlServiceKey

Sets a UDDI key for the native REST service.

Syntax

```
XMLServiceManager.deleteXMLService(String xmlServiceKey)
```

Parameters

String xmlServiceKey
UDDI key of the native REST service.

Importing a BPEL Process File

The BPEL importer imports objects of a Business Process Execution Language file. In CentraSite there are various specific predefined BPEL-ObjectTypes. A BPEL process flow usually references certain web services. Note that those web services should be registered prior to the BPEL registration; otherwise the references to the services cannot be established. The top-level controlling object is a BPELProcess object.

The following example demonstrates how to import a BPEL file:

```
import com.centrasite.jaxr.JAXRAccessor;
import com.centrasite.jaxr.bpel.BPELRegistrator;

String dbURL = "http://localhost:53307/CentraSite/CentraSite";
String bpelFile = "c:/temp/MyBPEL.bpel";
String user = ...;
String password = ...;

JAXRAccessor jaxr = new JAXRAccessor(dbURL, user, password);

try
{
   BPELRegistrator br = new BPELRegistrator(bpelFile, jaxr);
   br.register();
   int warnings = br.getWarningCount();    // are there unreferenced services?
}
catch (Exception e)
{
   // handle error
}
finally
{
```

```
    jaxr.close();
}
```

The `BPELRegistrator` class provides the following optional setter methods. If you want to use them, they must be called before calling the `register()` method.

setProjectName

Specifies a project name, i.e. an internal RegistryPackage. The named project will receive the created top-level object (a BPELProcess) as a member.

Syntax

```
public void setProjectName(String projectName)
```

Parameters

String projectName
The desired name of the project.

setWarningIfPLTNotFound

Specifies how CentraSite should react if a service that has not yet been registered in CentraSite is encountered.

Syntax

```
public void setWarningIfPLTNotFound(boolean warningIfPLTNotFound)
```

Parameters

boolean warningIfPLTNotFound

true
Each time a service that has not yet been registered in CentraSite is encountered, a counter is incremented. Use the `getWarningCount()` method to get the current value of the counter.

false
If a service that has not yet been registered in CentraSite is encountered, an exception is thrown.

Usage Notes

The default value of the parameter is "true".

Removing a Registered BPEL Object

You can remove a registered BPEL process object. The BPEL file is removed from the repository, and all associated registry objects are also removed. Like the `import` class, you must first establish a CentraSite connection with an instance of the `JAXRAccessor` class before calling the `remove` or `removeProcess` method. The `BPELAdministrator` class provides the following methods:

remove

Removes the BPEL objects of the process flow indicated in the specified BPEL file.

Syntax

```
public boolean remove (String bpelFile)
```

Parameters

String bpelFile
Specifies the BPEL file whose objects are to be removed. This could be the file in the CentraSite repository.

Return Codes

Value	Meaning
true	The BPEL objects were successfully removed.
false	The BPEL objects could not be removed.

removeProcess

Removes the BPEL objects of the process flow indicated by the BPELProcess name and its namespace.

Syntax

```
public boolean removeProcess(String bpelProcessName, String bpelNamespace
```

Parameters

String bpelProcessName
The process name of the BPEL objects to be removed.

String bpelNamespace
The namespace of the BPEL objects to be removed.

Return Codes

Value	Meaning
true	The BPEL objects were successfully removed.
false	The BPEL objects could not be removed.

Importing an XPDL File

The XPDL importer imports a process definition from an XPDL file. From the XPDL file, the importer produces a Process object and related components (e.g., Process Steps, Process Pools and Process Swimlanes). If the XPDL process references a Web service, the importer will add the Web service to the registry if it is not already present and associate it with the Process object.

The following example demonstrates how to import an XPDL file into CentraSite using the CentraSite Java API:

```
import com.centrasite.jaxr.JAXRAccessor;
import com.centrasite.jaxr.xpdl.ImportXPDL;

// Set URL for CentraSite Registry/Repository
String dbURL = "http://localhost:53307/CentraSite/CentraSite";

// Specify the location of the XPDL file.
String xpdlFile = "c:/temp/MyXPDL.xpdl";

// Specify the user account and password that the importer will use to log on to ↩
CentraSite
```

```
String user = "ctambrose";
String password = "jm6A1999";

// Build the connection to CentraSite
JAXRAccessor jaxr = new JAXRAccessor(dbURL, user, password);

try
{
// Instantiate XPDL importer object with specified XPDL file and connection info
// and then execute the import method.

   ImportXPDL xpdl = new ImportXPDL(xpdlFile, jaxr);
   xpdl.doImport();
}
catch (Exception e)
{
// Handle error
   ...
}
finally
{
 // Close connection to CentraSite Registry/Repository

   jaxr.close();
}
```

The `ImportXPDL` class provides the following optional setter methods that you can use to specify certain properties in the Process object. If you want to use these setter methods, you must call them before you call the `doImport()` method.

setOrganization

Specifies the organization to which the Process object is to be added.

Syntax

```
public void setOrganization (Organization org)
```

Parameters

Organization org

The organization to which the importer will add the Process object. Note that this method takes an Organization object as input. You can obtain the Organization object for a specified organization using the BusinessQueryManager.getRegistryObject() method.

Usage Notes

Optional. If you do not set the organization parameter, the importer will add the Process object to the organization that the user specified in the JAXRAccessor belongs.

setOriginalFilename

Specifies the filename to be assigned to the XPDL file in CentraSite's repository.

Syntax

```
public void setOriginalFilename (String originalFilename)
```

Parameters

String originalFilename
The filename that is to be given to the XPDL file in the CentraSite repository.

Note: A valid filename can consist of letters, numbers, and the underscore character. It must not contain spaces or other special characters.

Usage Notes

Optional.

setProductConcept

Specifies the category from the Product taxonomy by which the Process object is to be classified.

Syntax

```
public void setProductConcept(Concept product)
```

Parameters

Concept product

The Product category (concept) that is to be assigned to the Process object. This classification is generally used to identify the product from which the Process was published.

The following are some of the predefined categories in the Product taxonomy in CentraSite. (For other categories, examine the Product taxonomy on the instance of CentraSite to which you are importing the XPDL file.)

- CentraSite
- webMethods ApplinX
- webMethods EntireX
- webMethods Composite Application Framework
- webMethods Product Suite
  - webMethods BPM
  - webMethods Trading Networks
  - webMethods ESB

Usage Notes

Optional.

setVersion

Specifies the version identifier that is to be assigned to the Process object that the importer adds to the registry. (This methods specifies the user-defined *version identifier*. The Process object will also have a *version number*, which is automatically assigned and maintained by CentraSite.)

Syntax

```
public void setVersion(String version)
```

Parameters

String version
The version identifier to be assigned to the Process object.

Usage Notes

Optional.

2 Invoking an Importer from a Java Class

Each importer includes a `main()` method, which allows it to be called from a Windows batch file or from a UNIX shell script.

To invoke an importer from the command line, you must perform the following high-level steps:

1. Create a script file as described in ***Creating a Script File to Invoke an Importer***.
2. Execute the script file with the appropriate input parameters as described in ***Executing the Script File that Invokes an Importer***.

Creating a Script File to Invoke an Importer

The importers are Java classes whose `main()` method executes when you run the importer class from the command line. To ensure that the CLASSPATH and other environment variables are set properly when you run the importer class, you must create a script file as described in the following sections:

- Creating a Script File for Windows (a .bat file)
- Creating a Script File for Unix (C-shell script)

Creating a Script File for Windows (a .bat file)

Create a script file that looks as follows if CentraSite is running under Windows.

```
@echo off
set JAVAEXE=fullPathToJava.exe
set REDIST=CentraSiteHomeDirectory\redist
set BASEDIR=%~dp0
cd /d %REDIST%

REM build CLASSPATH with all files from jar directory
set LOCAL_CLASSPATH=
for %%I in (".\*.jar") do call "CentraSiteHomeDirectory\bin\cfg\lcp.cmd" %%I

%JAVAEXE% -cp %LOCAL_CLASSPATH% importerClassName %*
cd /d %BASEDIR%
```

Where *importerClassName* is the name of the Importer class that you want to run. See ***Importer Class Names*** for a list of the importer class names.

Example

The following is an example of a script file that calls the XML Schema importer:

```
@echo off
REM
REM Run XML Schema Importer
REM
set JAVAEXE=D:\software\java\jdk1.5.0_12\bin\java
set REDIST=C:\SoftwareAG\CentraSite\redist
set BASEDIR=%~dp0
cd /d %REDIST%

REM build CLASSPATH with all files from jar directory
set LOCAL_CLASSPATH=
for %%I in (".\*.jar") do call "C:\SoftwareAG\CentraSite\bin\cfg\lcp.cmd" %%I

%JAVAEXE% -cp %LOCAL_CLASSPATH% com.centrasite.jaxr.schema.SchemaImporter %*
cd /d %BASEDIR%
```

Creating a Script File for Unix (C-shell script)

Create a script file that looks as follows if CentraSite is running under Unix.

```
set javaexe="fullPathToJava.exe"
set redist="CentraSiteHomeDirectory/redist"
set mainjar="CentraSiteUtils.jar"
set delim='\:'
cd "$redist"
set cl=""
foreach j ( 'ls *.jar' )
  if ($cl != "") set cl=${cl}${delim}
  set cl=${cl}${j}
end
setenv CLASSPATH ${mainjar}${delim}${cl}
$javaexe importerClassName $*
```

Where *importerClassName* is the name of the Importer class that you want to run. See ***Importer Class Names*** for a list of the importer class names.

Example

The following is an example of a script file that calls the XML Schema importer:

```
#!/bin/csh
#
# Run XML Schema Importer
#
set javaexe="/mydir/softwareag/cjp/v16/bin/java"
set redist="/mydir/softwareag/CentraSite/redist"
set mainjar="CentraSiteUtils.jar"
set delim='\:'
# build CLASSPATH with all files from jar directory
```

```
cd "$redist"
set cl=""
foreach j ( 'ls *.jar' )
  if ($cl != "") set cl=${cl}${delim}
  set cl=${cl}${j}
end
setenv CLASSPATH ${mainjar}${delim}${cl}
$javaexe com.centrasite.jaxr.schema.SchemaImporter $*
```

Importer Class Names

The following are the class names for the importers:

Note: Some importers have an *import class*, which you use to import the asset, and an *admin utility class*, which you can use to delete or replace the asset after it has been imported.

Importer or Admin Utility that You Want to Use	Class Name
Web Service (Importer)	`com.centrasite.jaxr.webservice.WebServiceRegistrator`
Web Service (Admin utility)	`com.centrasite.jaxr.webservice.WebServiceAdministrator`
XML Service (Importer)	`com.centrasite.jaxr.xmlservice.XMLServiceManager`
REST Service (Importer)	`com.centrasite.jaxr.xmlservice.XMLServiceManager`
XML Schema (Importer)	`com.centrasite.jaxr.schema.SchemaImporter`
XML Schema (Admin utility)	`com.centrasite.jaxr.schema.SchemaAdministrator`
BPEL Process Definition (Importer)	`com.centrasite.jaxr.bpel.BPELRegistrator`
BPEL Process Definition (Admin utility)	`com.centrasite.jaxr.bpel.BPELAdministrator`
XPDL File (Importer)	`com.centrasite.jaxr.xpdl.ImportXPDL`

Executing the Script File that Invokes an Importer

To invoke the importer, you must run the importer script file with the required set of input parameters. (If you need information about creating the script file, see ***Creating a Script File to Invoke an Importer***.)

The input parameters that are required to run the script file will vary depending on the importer your script file invokes. The input parameters for each importer are described in the following sections:

- Importing a Web Service
- Importing an XML Schema
- Importing an XML Service

- Importing a REST Service
- Importing a BPEL Process
- Importing an XPDL File

Importing a Web Service

To import a Web service, run your importer script file with the following required input parameters:

```
yourScriptFile -w wsdlFile -o orgName -user yourCSUserID -password yourPassword
```

Example

```
myScript -w d:\myDirectory\myWSDLFile.wsdl -o "Customer Service" -user jcambrose
-password j45Hk19a
```

Input Parameters

The following table describes the complete set of input parameters that you can use with the Web Service importer:

Parameter	Description
`-w wsdlFile`	*Required*. The fully qualified name of the WSDL file that you want to import.
`-o orgName`	*Required*. The name of the organization into which you want to import the Web service. If the organization name contains a space, enclose the name in double-quotes.
`-user yourCSUserID`	*Required*. Your CentraSite user ID.
`-password password`	*Required*. The password for your CentraSite user account.
`-h hostName`	The host name or IP address of the computer where the CentraSite registry/repository component is running. If you omit this parameter, the importer assumes that the registry/repository is running on `localhost`.
`-p portNumber`	The port number on which the CentraSite registry/repository is configured to listen for incoming requests. If you omit this parameter, the importer assumes that the registry/repository is listening on the default port, `53307`.
`-dburl url`	The fully qualified URL for the CentraSite registry/repository. If you omit this parameter, the importer assumes that the registry/repository resides at `http://localhost:53307/CentraSite/CentraSite`.

Note: You can also obtain the complete list of input parameters for the Web Service importer by invoking the importer with no input parameters.

Invoking the Web Service Administrator from the Command Line

You can invoke the Web Service Administrator utility to delete a Web service that has been imported into CentraSite. To obtain the input parameters used by this utility, run the script file with no input parameters.

Importing an XML Schema

To import an XML Schema, run your importer script file with the following input parameters:

```
yourScriptFile -s xsdFile -user yourCSUserID -password yourPassword
```

Example

```
myScript -s d:\myDirectory\myXSDFile.xsd -user jcambrose -password j45Hk19a
```

Input Parameters

The following table describes the complete set of input parameters that you can use with the XML Schema importer:

Parameter	Description
`-s xsdFile`	*Required*. The fully qualified name of the XML Schema file that you want to import.
`-user user ID`	*Required*. Your CentraSite user ID.
`-password password`	*Required*. The password for your CentraSite user account.
`-h hostName`	The host name or IP address of the computer where the CentraSite registry/repository component is running. If you omit this parameter, the importer assumes that the registry/repository is running on `localhost`.
`-p portNumber`	The port number on which the CentraSite registry/repository is configured to listen for incoming requests. If you omit this parameter, the importer assumes that the registry/repository is listening on the default port, `53307`.
`-dburl url`	The fully qualified URL for the CentraSite registry/repository. If you omit this parameter, the importer assumes that the registry/repository resides at `http://localhost:53307/CentraSite/CentraSite`. **Note:** If the registry/repository is running on a different machine and port number, you can use this parameter to specify its location instead of using the individual -h and -p parameters. (If you specify the -dburl parameter with the -h and/or -p parameters, the -h and -p parameters will be ignored.)
`-noover`	Prevents the importer from overwriting a schema if it is already present in the registry.

Note: You can also obtain the complete list of input parameters for the XML Schema importer by invoking the importer with no input parameters.

Invoking the XML Schema Administrator from the Command Line

You can invoke the XML Schema Administrator utility to delete an XML Schema that has been imported into CentraSite. To obtain the input parameters used by this utility, run the script file with no input parameters.

Importing an XML Service

To import an XML Service, run your importer script file with the following input parameters:

yourScriptFile -s *xmlFileURI* -n *serviceName* -e *endpointURL* -m *httpMethods* -dbuser *yourCSUserID* -dbpassword *yourPassword*

Example

```
myScript -s http://fs02hq/xml/myService.xsd -n myXMLService -e
http://appsvr02:53307/myService -m GET PUT -dbuser jcambrose -dbpassword j45Hk19a
```

Input Parameters

The following table describes the complete set of input parameters that you can use with the XML Service importer:

Parameter	Description
-s *xsdFile*	*Required*. The URI (file: or http:) of the XML Schema file from which the XML service is created.
-n *serviceName*	*Required*. The name that is to be assigned to the service.
-e *endpointURL*	*Required*. The URL where the service is deployed. (The access point for the service.)
-m *httpMethods*	*Required*. The HTTP access methods that the service supports. Valid values are: GET, POST, PUT, DELETE. If the service supports multiple methods, list the methods separated by spaces (e.g., `-m GET PUT DELETE`).
-dbuser *user ID*	*Required*. Your CentraSite user ID. (This user account must have permission to create assets in the organization to which the service will be added.)
-dbpassword *password*	*Required*. The password for your CentraSite user account.
-dbhost *hostName*	The host name or IP address of the computer where the CentraSite registry/repository component is running. If you omit this parameter, the importer assumes that the registry/repository is running on `localhost`.
-dbport *portNumber*	The port number on which the CentraSite registry/repository is configured to listen for incoming requests.

Parameter	Description
	If you omit this parameter, the importer assumes that the registry/repository is listening on the default port, 53307.
-suser *xsdFileUser*	The user ID that is to be used if the XSD file (specified in the -s parameter) resides on a secure server and the importer is required to provide a user ID and password to access it.
-spassword *xsdFilePassword*	The password for the user account in -suser.
-noover	Do not overwrite existing imported XML Schemas that exist in the registry.
-desc *description*	The description that is to be assigned to the XML Service. If you omit this parameter, the service description will be empty.
-ver *versionID*	The version identifier that is to be assigned to the XML Service. (This is the user-defined identifier, not the system-assigned version number.) If you omit this parameter, the version identifier will be set to 1.0.
-porg *providingOrg*	The name of the organization that is to be designated as the providing organization for this service. If you omit this parameter, the providing organization will be set to the Default Organization.
-sorg *submittingOrg*	The name of the organization to which this service is to be added. If you omit this parameter, the service will be added to the Default Organization.

Note: You can also obtain the complete list of input parameters for the XML Service importer by invoking the importer with no input parameters.

Importing a REST Service

To import a REST Service, run your importer script file with the following input parameters:

yourScriptFile -s *xmlFileURI* -n *serviceName* -e *endpointURL* -m *httpMethods* -dbuser *yourCSUserID* -dbpassword *yourPassword*

Example

```
myScript -s http://fs02hq/xml/myService.xsd -n myXMLService -e
http://appsvr02:53307/myService -m GET PUT -dbuser jcambrose -dbpassword j45Hk19a
```

Input Parameters

The following table describes the complete set of input parameters that you can use with the REST Service importer:

Parameter	Description
`-s` *`xsdFile`*	*Required*. The URI (file: or http:) of the XML Schema file from which the REST service is created.
`-n` *`serviceName`*	*Required*. The name that is to be assigned to the service.
`-e` *`endpointURL`*	*Required*. The URL where the service is deployed. (The access point for the service.)
`-m` *`httpMethods`*	*Required*. The HTTP access methods that the service supports. Valid values are: GET, POST, PUT, DELETE. If the service supports multiple methods, list the methods separated by spaces (e.g., `-m GET PUT DELETE`).
`-dbuser` *`user ID`*	*Required*. Your CentraSite user ID. (This user account must have permission to create assets in the organization to which the service will be added.)
`-dbpassword` *`password`*	*Required*. The password for your CentraSite user account.
`-dbhost` *`hostName`*	The host name or IP address of the computer where the CentraSite registry/repository component is running. If you omit this parameter, the importer assumes that the registry/repository is running on `localhost`.
`-dbport` *`portNumber`*	The port number on which the CentraSite registry/repository is configured to listen for incoming requests. If you omit this parameter, the importer assumes that the registry/repository is listening on the default port, `53307`.
`-suser` *`xsdFileUser`*	The user ID that is to be used if the XSD file (specified in the -s parameter) resides on a secure server and the importer is required to provide a user ID and password to access it.
`-spassword` *`xsdFilePassword`*	The password for the user account in `-suser`.
`-noover`	Do not overwrite existing imported XML Schemas that exist in the registry.
`-desc` *`description`*	The description that is to be assigned to the REST Service. If you omit this parameter, the service description will be empty.
`-ver` *`versionID`*	The version identifier that is to be assigned to the REST Service. (This is the user-defined identifier, not the system-assigned version number.) If you omit this parameter, the version identifier will be set to 1.0.
`-porg` *`providingOrg`*	The name of the organization that is to be designated as the providing organization for this service. If you omit this parameter, the providing organization will be set to the Default Organization.

Parameter	Description
`-sorg` *submittingOrg*	The name of the organization to which this service is to be added. If you omit this parameter, the service will be added to the Default Organization.

Note: You can also obtain the complete list of input parameters for the REST Service importer by invoking the importer with no input parameters.

Importing a BPEL Process

To import a BPEL process, call your importer script file with the following input parameters:

yourScriptFile `-file` *bpelFile* `-user` *yourCSUserID* `-password` *yourPassword*

Example

```
myScript -file d:\myDirectory\myBPELFile.bpel -user jcambrose -password j45Hk19a
```

Input Parameters

The following table describes the complete set of input parameters that you can use with the BPEL importer:

Parameter	Description
`-file` *bpelFile*	*Required*. The fully qualified name of the BPEL process file that you want to import.
`-user` *user ID*	*Required*. Your CentraSite user ID.
`-password` *password*	*Required*. The password for your CentraSite user account.
`-h` *hostName*	*Optional.* The host name or IP address of the computer where the CentraSite registry/repository component is running. If you omit this parameter, the importer assumes that the registry/repository is running on `localhost`.
`-p` *portNumber*	The port number on which the CentraSite registry/repository is configured to listen for incoming requests. If you omit this parameter, the importer assumes that the registry/repository is listening on the default port, `53307`.
`-dburl` *url*	The fully qualified URL for the CentraSite registry/repository. If you omit this parameter, the importer assumes that the registry/repository resides at `http://localhost:53307/CentraSite/CentraSite`. **Note:** If the registry/repository is running on a different machine and port number, you can use this parameter to specify its location instead of using the individual -h

Parameter	Description
	and -p parameters. (If you specify the -dburl parameter with the -h and/or -p parameters, the -h and -p parameters will be ignored.)
`-nowarning`	Suppresses the warning message that the importer issues if the BPEL process references a BPEL Partner Link Type that refers to a Web service that is not present in the CentraSite registry.

Note: You can also obtain the complete list of input parameters for the BPEL importer by invoking the importer with no input parameters.

Invoking the BPEL Administrator from the Command Line

You can invoke the BPEL Administrator utility to delete or display a BPEL process that has been imported into CentraSite. To obtain the input parameters used by this utility, invoke the BPEL administrator with no input parameters.

Importing an XPDL File

To import an XPDL file, call the importer script file from the command line as shown below.

```
yourScriptFile -file xpdlFile -user yourCSUserID -password yourPassword
```

Note: For additional input parameters that you can use with this importer, see ***Input Parameters*** below.

Example

```
myScript -file d:\myDirectory\myXPDLFile.xpdl -user jcambrose -password j45Hk19a
```

Input Parameters

The following table describes input parameters that you can use with the XPDL importer:

Parameter	Description
`-file` *`xpdlFile`*	*Required*. The fully qualified name of the XPDL file that you want to import.
`-user` *`user ID`*	*Required*. Your CentraSite user ID.
`-password` *`password`*	*Required*. The password for your CentraSite user account.
`-h` *`hostName`*	The host name or IP address of the computer where the CentraSite registry/repository component is running. If you omit this parameter, the importer assumes that the registry/repository is running on `localhost`.

Parameter	Description
`-p portNumber`	The port number on which the CentraSite registry/repository is configured to listen for incoming requests. If you omit this parameter, the importer assumes that the registry/repository is listening on the default port, `53307`.

Note: You can also obtain the complete list of input parameters for the XPDL importer by calling the script file with no input parameters.

3 Invoking an Importer From the Command Line

Each importer includes a *command*, which allows it to be executed from the command prompt on Windows or a terminal on any UNIX flavor.

In both cases, the command is followed by a list of appropriate input parameters and options. The parameters are key-value pairs, where the key starts with a hyphen. An option is a switch that modifies the behavior of the command and is preceded by a hyphen. Using the -usage option lists all available command parameters and options with a short description.

The input parameters and options for each importer are described in the following sections:

Importing a Web Service

To import a Web service, run the utility with the following input parameters:

- Command Line under Windows
- Command Line under Linux
- Input Parameters

Command Line under Windows

Use the following procedure to import a Web service under Windows:

1. Open *ImportWSDL.cmd* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>\utilities\ImportWSDL.cmd -user <USERNAME> -password ↩
<PASSWORD> [-dburl <CENTRASITE-URL> | [[-h <HOSTNAME>] [-p <PORT>]]] -o ↩
<ORGANIZATION> -w <WSDL-FILE-PATH>
```

where, <*CentraSiteInstallDir*> is the CentraSite installation directory. By default, this is the *CentraSite* folder under <*SuiteInstallDir*>.

Command Line under Linux

Use the following procedure to import a Web service under Linux:

1. Open *ImportWSDL.sh* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>\utilities\ImportWSDL.sh -user <USERNAME> -password ↩
<PASSWORD> [-dburl <CENTRASITE-URL> | [[-h <HOSTNAME>] [-p <PORT>]]] -o ↩
<ORGANIZATION> -w <WSDL-FILE-PATH>
```

where, *<CentraSiteInstallDir>* is the CentraSite installation directory. By default, this is the *CentraSite* folder under *<SuiteInstallDir>*.

Input Parameters

The following table describes the complete set of input parameters that you can use with the Web Service importer:

Parameter	Description
USERNAME	*Required*. Your CentraSite user ID.
PASSWORD	*Required*. The password for your CentraSite user account.
CENTRASITE-URL	The fully qualified URL for the CentraSite registry/repository. If you omit this parameter, the importer assumes that the registry/repository resides at `http://localhost:53307/CentraSite/CentraSite`. **Note:** If the registry/repository is running on a different machine and port number, you can use this parameter to specify its location instead of using the individual -h and -p parameters. (If you specify the -dburl parameter with the -h and/or -p parameters, the -h and -p parameters will be ignored.)
HOSTNAME	The host name or IP address of the computer where the CentraSite registry/repository component is running. If you omit this parameter, the importer assumes that the registry/repository is running on `localhost`.
PORT	The port number on which the CentraSite registry/repository is configured to listen for incoming requests. If you omit this parameter, the importer assumes that the registry/repository is listening on the default port, `53307`.
ORGANIZATION	*Required*. The name of the organization into which you want to import the Web service. If the organization name contains a space, enclose the name in double-quotes.
WSDL-FILE-PATH	*Required*. The fully qualified name of the WSDL file that you want to import.

Note: You can also obtain the complete list of input parameters for the Web Service importer by invoking the importer with no input parameters.

Importing an XML Schema

To import an XML Schema, run the utility with the following input parameters and options:

Command Line under Windows

Use the following procedure to import an XML Schema under Windows:

1. Open *ImportSchema.cmd* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>\utilities\ImportSchema.cmd -user <USERNAME> -password ↩
<PASSWORD> [-dburl <CENTRASITE-URL> | [[-h <HOSTNAME>] [-p <PORT>]]] -s <XSD-FILE> ↩
[-noover]
```

where, *<CentraSiteInstallDir>* is the CentraSite installation directory. By default, this is the *CentraSite* folder under *<SuiteInstallDir>*.

Command Line under Linux

Use the following procedure to import an XML Schema under Linux:

1. Open *ImportSchema.sh* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>\utilities\ImportSchema.sh -user <USERNAME> -password ↩
<PASSWORD> [-dburl <CENTRASITE-URL> | [[-h <HOSTNAME>] [-p <PORT>]]] -s <XSD-FILE> ↩
[-noover]
```

where, *<CentraSiteInstallDir>* is the CentraSite installation directory. By default, this is the *CentraDESTINATIONSite* folder under *<SuiteInstallDir>*.

Input Parameters and Options

The following tables describe the complete set of input parameters and options that you can use with the XML schema importer:

Parameter	Description
USERNAME	*Required*. Your CentraSite user ID.
PASSWORD	*Required*. The password for your CentraSite user account.
CENTRASITE-URL	The fully qualified URL for the CentraSite registry/repository. If you omit this parameter, the importer assumes that the registry/repository resides at `http://localhost:53307/CentraSite/CentraSite`. **Note:** If the registry/repository is running on a different machine and port number, you can use this parameter to specify its location instead of using the individual -h and -p parameters. (If you specify the -dburl parameter with the -h and/or -p parameters, the -h and -p parameters will be ignored.)
HOSTNAME	The host name or IP address of the computer where the CentraSite registry/repository component is running. If you omit this parameter, the importer assumes that the registry/repository is running on `localhost`.
PORT	The port number on which the CentraSite registry/repository is configured to listen for incoming requests. If you omit this parameter, the importer assumes that the registry/repository is listening on the default port, `53307`.
XSD-FILE	*Required*. The fully qualified name of the XML Schema file that you want to import.

Option	Description
`-noover`	Prevents the importer from overwriting an XML schema if it is already present in the registry.

Note: You can also obtain the complete list of input parameters for the XML schema importer by invoking the importer with no input parameters.

Importing an XML Service

To import an XML Service, run the utility with the following input parameters and options:

- Command Line under Windows
- Command Line under Linux

Command Line under Windows

Use the following procedure to import an XML Service under Windows:

1. Open *ImportXMLService.cmd* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>\utilities\ImportXMLService.cmd -dbuser <USERNAME> ↩
-dbpassword <PASSWORD> [[-h <HOSTNAME>] [-p <PORT>]] -s <XML-FILE> -n ↩
<SERVICE-NAME> -e <ENDPOINT-URL> -m <HTTP-METHODS> [[-suser <XML-FILE-USER>] ↩
[-spassword <XML-FILE-PASSWORD>]] [-noover] [-desc <DESCRIPTION>] [-ver ↩
<VERSION-ID>] [-porg <PROVIDING-ORG>] [-sorg <SUBMITTING-ORG>]
```

where, `<CentraSiteInstallDir>` is the CentraSite installation directory. By default, this is the *CentraSite* folder under `<SuiteInstallDir>`.

Command Line under Linux

Use the following procedure to import an XML Service under Linux:

1. Open *ImportXMLService.sh* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>\utilities\ImportXMLService.sh -dbuser <USERNAME> ↩
-dbpassword <PASSWORD> [[-h <HOSTNAME>] [-p <PORT>]] -s <XML-FILE> -n ↩
<SERVICE-NAME> -e <ENDPOINT-URL> -m <HTTP-METHODS> [[-suser <XML-FILE-USER>] ↩
[-spassword <XML-FILE-PASSWORD>]] [-noover] [-desc <DESCRIPTION>] [-ver ↩
<VERSION-ID>] [-porg <PROVIDING-ORG>] [-sorg <SUBMITTING-ORG>]
```

where, `<CentraSiteInstallDir>` is the CentraSite installation directory. By default, this is the *CentraSite* folder under `<SuiteInstallDir>`.

Input Parameters and Options

The following tables describe the complete set of input parameters and options that you can use with the XML Service importer:

Parameter	Description
XML-FILE	*Required*. The URI (file: or http:) of the XML Schema file from which the XML service is created.
SERVICE-NAME	*Required*. The name that is to be assigned to the service.
ENDPOINT-URL	*Required*. The URL where the service is deployed. (The access point for the service.)
HTTP-METHODS	*Required*. The HTTP access methods that the service supports. Valid values are: GET, POST, PUT, DELETE. If the service supports multiple methods, list the methods separated by spaces (e.g., `-m GET PUT DELETE`).
USERNAME	*Required*. Your CentraSite user ID. (This user account must have permission to create assets in the organization to which the service will be added.)
PASSWORD	*Required*. The password for your CentraSite user account.
HOSTNAME	The host name or IP address of the computer where the CentraSite registry/repository component is running. If you omit this parameter, the importer assumes that the registry/repository is running on `localhost`.
PORT	The port number on which the CentraSite registry/repository is configured to listen for incoming requests. If you omit this parameter, the importer assumes that the registry/repository is listening on the default port, `53307`.
XML-FILE-USER	The user ID that is to be used if the XML file (specified in the -s parameter) resides on a secure server and the importer is required to provide a user ID and password to access it.
XML-FILE-PASSWORD	The password for the user account in `-suser`.
DESCRIPTION	The description that is to be assigned to the XML Service. If you omit this parameter, the service description will be empty.
VERSION-ID	The version identifier that is to be assigned to the XML Service. (This is the user-defined identifier, not the system-assigned version number.) If you omit this parameter, the version identifier will be set to 1.0.
PROVIDING-ORG	The name of the organization that is to be designated as the providing organization for this service. If you omit this parameter, the providing organization will be set to the Default Organization.
SUBMITTING-ORG	The name of the organization to which this service is to be added. If you omit this parameter, the service will be added to the Default Organization.

Option	Description
`-noover`	Prevents the importer from overwriting an XML service if it is already present in the registry.

Note: You can also obtain the complete list of input parameters for the XML Service importer by invoking the importer with no input parameters.

Importing a REST Service

To import a REST Service, run the utility with the following input parameters and options:

- Command Line under Windows
- Command Line under Linux
- Input Parameters and Options

Command Line under Windows

Use the following procedure to import a REST Service under Windows:

1. Open *ImportRESTService.cmd* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>\utilities\ImportRESTService.cmd -dbuser <USERNAME> ↩
-dbpassword <PASSWORD> [[-h <HOSTNAME>] [-p <PORT>]] -s <REST-FILE> -n ↩
<SERVICE-NAME> -e <ENDPOINT-URL> -m <HTTP-METHODS> [[-suser <REST-FILE-USER>] ↩
[-spassword <REST-FILE-PASSWORD>]] [-noover] [-desc <DESCRIPTION>] [-ver ↩
<VERSION-ID>] [-porg <PROVIDING-ORG>] [-sorg <SUBMITTING-ORG>]
```

where, `<CentraSiteInstallDir>` is the CentraSite installation directory. By default, this is the *CentraSite* folder under `<SuiteInstallDir>`.

Command Line under Linux

Use the following procedure to import a REST Service under Linux:

1. Open *ImportRESTService.sh* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>\utilities\ImportRESTService.sh -dbuser <USERNAME> ↩
-dbpassword <PASSWORD> [[-h <HOSTNAME>] [-p <PORT>]] -s <REST-FILE> -n ↩
<SERVICE-NAME> -e <ENDPOINT-URL> -m <HTTP-METHODS> [[-suser <REST-FILE-USER>] ↩
[-spassword <REST-FILE-PASSWORD>]] [-noover] [-desc <DESCRIPTION>] [-ver ↩
<VERSION-ID>] [-porg <PROVIDING-ORG>] [-sorg <SUBMITTING-ORG>]
```

where, *<CentraSiteInstallDir>* is the CentraSite installation directory. By default, this is the *CentraSite* folder under *<SuiteInstallDir>*.

Input Parameters and Options

The following tables describe the complete set of input parameters and options that you can use with the REST Service importer:

Parameter	Description
REST-FILE	*Required*. The URI (file: or http:) of the XML Schema file from which the REST service is created.
SERVICE-NAME	*Required*. The name that is to be assigned to the service.
ENDPOINT-URL	*Required*. The URL where the service is deployed. (The access point for the service.)
HTTP-METHODS	*Required*. The HTTP access methods that the service supports. Valid values are: GET, POST, PUT, DELETE. If the service supports multiple methods, list the methods separated by spaces (e.g., `-m GET PUT DELETE`).
USERNAME	*Required*. Your CentraSite user ID. (This user account must have permission to create assets in the organization to which the service will be added.)
PASSWORD	*Required*. The password for your CentraSite user account.
HOSTNAME	The host name or IP address of the computer where the CentraSite registry/repository component is running. If you omit this parameter, the importer assumes that the registry/repository is running on `localhost`.
PORT	The port number on which the CentraSite registry/repository is configured to listen for incoming requests. If you omit this parameter, the importer assumes that the registry/repository is listening on the default port, `53307`.
REST-FILE-USER	The user ID that is to be used if the REST file (specified in the -s parameter) resides on a secure server and the importer is required to provide a user ID and password to access it.
REST-FILE-PASSWORD	The password for the user account in `-suser`.
DESCRIPTION	The description that is to be assigned to the REST Service. If you omit this parameter, the service description will be empty.
VERSION-ID	The version identifier that is to be assigned to the REST Service. (This is the user-defined identifier, not the system-assigned version number.)

Parameter	Description
	If you omit this parameter, the version identifier will be set to 1.0.
PROVIDING-ORG	The name of the organization that is to be designated as the providing organization for this service. If you omit this parameter, the providing organization will be set to the Default Organization.
SUBMITTING-ORG	The name of the organization to which this service is to be added. If you omit this parameter, the service will be added to the Default Organization.

Option	Description
`-noover`	Prevents the importer from overwriting a REST service if it is already present in the registry.

Note: You can also obtain the complete list of input parameters for the REST Service importer by invoking the importer with no input parameters.

Importing a BPEL Process

To import a BPEL process, run the utility with the following input parameters and options:

- Command Line under Windows
- Command Line under Linux
- Input Parameters and Options

Command Line under Windows

Use the following procedure to import a BPEL process under Windows:

1. Open *ImportBPEL.cmd* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>\utilities\ImportBPEL.cmd -user <USERNAME> -password ↩
<PASSWORD> [[-dburl CENTRASITE-URL]|[[-h <HOSTNAME>] [-p <PORT>]]] -file ↩
<BPEL-FILE> [-nowarning]
```

where, *<CentraSiteInstallDir>* is the CentraSite installation directory. By default, this is the *CentraSite* folder under *<SuiteInstallDir>*.

Command Line under Linux

Use the following procedure to import a BPEL process under Linux:

1. Open *ImportBPEL.sh* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>\utilities\ImportBPEL.sh -user <USERNAME> -password ↩
<PASSWORD> [[-dburl CENTRASITE-URL]|[[-h <HOSTNAME>] [-p <PORT>]]] -file ↩
<BPEL-FILE> [-nowarning]
```

where, *<CentraSiteInstallDir>* is the CentraSite installation directory. By default, this is the *CentraSite* folder under *<SuiteInstallDir>*.

Input Parameters and Options

The following tables describe the complete set of input parameters and options that you can use with the BPEL Process importer:

Parameter	Description
BPEL-FILE	*Required*. The fully qualified name of the BPEL process file that you want to import.
USERNAME	*Required*. Your CentraSite user ID.
PASSWORD	*Required*. The password for your CentraSite user account.
HOSTNAME	*Optional.* The host name or IP address of the computer where the CentraSite registry/repository component is running. If you omit this parameter, the importer assumes that the registry/repository is running on `localhost`.
PORT	The port number on which the CentraSite registry/repository is configured to listen for incoming requests. If you omit this parameter, the importer assumes that the registry/repository is listening on the default port, `53307`.
CENTRASITE-URL	The fully qualified URL for the CentraSite registry/repository. If you omit this parameter, the importer assumes that the registry/repository resides at `http://localhost:53307/CentraSite/CentraSite`. **Note:** If the registry/repository is running on a different machine and port number, you can use this parameter to specify its location instead of using the individual -h and -p parameters. (If you specify the -dburl parameter with the -h and/or -p parameters, the -h and -p parameters will be ignored.)

Option	Description
`-nowarning`	Suppresses the warning message that the importer issues if the BPEL process references a BPEL Partner Link Type that refers to a Web service that is not present in the CentraSite registry.

Note: You can also obtain the complete list of input parameters for the BPEL importer by invoking the importer with no input parameters.

Importing an XPDL File

To import an XPDL file, run the utility with the following input parameters:

- Command Line under Windows
- Command Line under Linux
- Input Parameters

Command Line under Windows

Use the following procedure to import an XPDL file under Windows:

1. Open *ImportXPDL.cmd* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>\utilities\ImportXPDL.cmd -user <USERNAME> -password ↩
<PASSWORD> [[-h <HOSTNAME>] [-p <PORT>]] -file <XPDL-FILE>
```

where, `<CentraSiteInstallDir>` is the CentraSite installation directory. By default, this is the *CentraSite* folder under `<SuiteInstallDir>`.

Command Line under Linux

Use the following procedure to import an XPDL file under Linux:

1. Open *ImportXPDL.sh* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>\utilities\ImportXPDL.sh -user <USERNAME> -password ↩
<PASSWORD> [[-h <HOSTNAME>] [-p <PORT>]] -file <XPDL-FILE>
```

where, *<CentraSiteInstallDir>* is the CentraSite installation directory. By default, this is the *CentraSite* folder under *<SuiteInstallDir>*.

Input Parameters

The following table describes the complete set of input parameters that you can use with the XPDL file importer:

Parameter	Description
XPDL-FILE	*Required*. The fully qualified name of the XPDL file that you want to import.
USERNAME	*Required*. Your CentraSite user ID.
PASSWORD	*Required*. The password for your CentraSite user account.
HOSTNAME	The host name or IP address of the computer where the CentraSite registry/repository component is running. If you omit this parameter, the importer assumes that the registry/repository is running on `localhost`.
PORT	The port number on which the CentraSite registry/repository is configured to listen for incoming requests. If you omit this parameter, the importer assumes that the registry/repository is listening on the default port, `53307`.

Note: You can also obtain the complete list of input parameters for the XPDL file importer by invoking the importer with no input parameters.

Importing an Archive

To import an archive, run the utility with the following input parameters and options:

- Command Line under Windows
- Command Line under Linux

Command Line under Windows

Use the following procedure to import an archive under Windows:

1. Open *ImportArchive.cmd* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>\utilities\ImportArchive.cmd [-setreplace]
[-importorg <ORGANIZATION-KEY>] [-keepowner] [-keeporganization] <CENTRASITE_URL>
<ARCHIVE_FILENAME> <USERNAME> <PASSWORD>
```

where, *<CentraSiteInstallDir>* is the CentraSite installation directory. By default, this is the *CentraSite* folder under *<SuiteInstallDir>*.

Command Line under Linux

Use the following procedure to import an archive under Linux:

1. Open *ImportArchive.sh* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>\utilities\ ImportArchive.sh [-setreplace]
[-importorg <ORGANIZATION-KEY>] [-keepowner] [-keeporganization] <CENTRASITE_URL>
<ARCHIVE_FILENAME> <USERNAME> <PASSWORD>
```

where, *<CentraSiteInstallDir>* is the CentraSite installation directory. By default, this is the *CentraSite* folder under *<SuiteInstallDir>*.

Input Parameters and Options

The following table describes the complete set of input parameters and options that you can use with the archive importer:

Parameter	Description
ARCHIVE_FILENAME	*Required*. The fully qualified name of the export archive file. The archive file can contain an organization with its assets or can contain a set of objects that were exported from one or more organizations.
USERNAME	*Required*. Your CentraSite user ID.
PASSWORD	*Required*. The password for your CentraSite user account.
CENTRASITE-URL	*Required*. The fully qualified URL for the CentraSite registry/repository. If you omit this parameter, the importer assumes that the registry/repository resides at `http://localhost:53307/CentraSite/CentraSite`.

Parameter	Description
	Note: If the registry/repository is running on a different machine and port number, you can use this parameter to specify its location instead of using the individual -h and -p parameters. (If you specify the -dburl parameter with the -h and/or -p parameters, the -h and -p parameters will be ignored.)
ORGANIZATION-KEY	By default, objects are imported into the current user's organization. Use this optional parameter if you want to specify a different organization into which the objects should be imported. The keyword is followed by the organization's GUID, prefixed by "uddi:", for example: "uddi:207ff1cc-25c5-544c-415c-5d98ea91060c". **Note:** This parameter has no effect if the `-keeporganization` option is set.

Option	Description
`-setreplace`	When an object is to be imported, the timestamp of the object in the import archive file is compared with the timestamp of the corresponding object with the same GUID in the registry, if the latter exists. If the timestamps are equal, the object is not imported. If the archive timestamp is older than the registry timestamp, the object is only imported if this optional parameter is present; otherwise it is ignored.
`-keepowner`	This option will maintain the original owner of the importing objects. This requires the original owner to exist with the same UDDI key on the target machine. If the parameter is not set, the importing user will be owner of the objects.
`-keeporganization`	This option will maintain the original organization of the importing objects. This requires the original organization to exist with the same UDDI key on the target machine. **Note:** If an organization is imported with its assets, this option is automatically activated.

Note: You can also obtain the complete list of input parameters for the XPDL file importer by invoking the importer with no input parameters.

4 Invoking an Importer Using the SOAP API

CentraSite provides a Web service for each of the predefined importers. Descriptions of these services are available here:

```
http://server:port/wsstack/services/listServices
```

where *server* is the machine on which the CentraSite Application Server Tier is running and *port* is the port on which Tomcat is listening (port 53307 if CentraSite is configured to use the default Tomcat port number).

Example

```
http://myServer:53307/wsstack/services/listServices
```

We recommend using MTOM (see the external W3C page "SOAP Message Transmission Optimization Mechanism" at **http://www.w3.org/TR/soap12-mtom/** for details) when using the CentraSite web services with attachments of 1 megabyte or more.

Viewing the WSDL for the Importers

To view the WSDL for an importer service, click the service name on the listServices page.

5 Writing Your Own Importer

You can write your own plug-in for CentraSite Control to incorporate your own importer. The prepared plug-in is a collection of files in a specific directory structure. After implementing the plug-in, the files must be copied into the CentraSite Control *webapps* folder under:

```
<RuntimeWebAppsDir>/PluggableUI/<MyPluginFolder>
```

(The location of the <RuntimeWebAppsDir> folder is described in the document *Basic Operations*).

The folder <MyPluginFolder> must contain the following files and folders:

Name of File or Folder	Description
plugin.xml	Top plug-in description file
*.html	Files generated for the GUI (see ***build.xml***)
*_SWT.xml	Files generated for the GUI (see ***build.xml***)
images	Directory for icons (GIF files)
lib	Directory for JAR files provided by the plug-in
accesspath	Directory created by HTML generation (see ***build.xml***)

In the following sections, we demonstrate a framework named ImportMyFile that illustrates how an import plug-in may be set up. The example extends the import selection list and presents a screen that prompts for the file to be imported. After confirming the file, the appropriate adapter classes are called.

You may use this example as a guideline, adapting it and renaming it to suit your individual requirements. The templates indicate where customization is required.

The following topics are discussed in this chapter:

The Build Environment

This section explains the build environment for generating the HTML files that are used for the GUI and for compiling the necessary Java source files. It assumes the use of Ant (**http://ant.apache.org/**), the Java-based build tool.

The following file system structure under the plug-in directory is assumed:

Name of File or Folder	Description
src	The directory that holds the Java source files
xml	The directory that holds the XML file that specifies your import window
plugin.xml	Top plugin description file that specifies the extension point and the command class
build.xml	The Ant input file for building the destination files

The Ant file shown below, named *build.xml*, can be used to establish an import plug-in. Look for the properties with the following names:

plugin.name
plugin.provider
tomcat.dir
tomcat.ver.dir

and modify them as required to suit your installation.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="./ant2html.xsl"?>
<!--
  Build an Import plugin
-->

<project name="Import Plugin Example" default="all" basedir=".">

  <description> Build file for an Importer plugin </description>

  <!-- environment -->
  <property environment="env"/>

  <property name="plugin.name" value="ImportMyFile" />
  <property name="plugin.provider" value="Software AG" />

  <!-- tomcat home directory -->
  <!-- <property name="tomcat.dir" value="tomcat directory of installation" /> -->
  <property name="tomcat.dir" value="C:/SoftwareAG/profiles/CTP" />
  <property name="tomcat.ver.dir" value="${tomcat.dir}/workspace"/>

  <!-- point to the root of the pluggableUI to get the cis environment -->
  <property name="pluggable.ui.root" value="${tomcat.ver.dir}/webapps/PluggableUI"/>

  <!-- the directory containing source code -->
  <property name="plugin.dir" value="../${plugin.name}" />
  <property name="src.dir"    value="${plugin.dir}/src" />
  <property name="xml.dir"    value="${plugin.dir}/xml" />
  <property name="classes"    value="${plugin.dir}/classes" />
  <property name="lib"        value="${plugin.dir}/lib" />

  <!-- classpath -->
```

```
  <path id="plugin.class.path">
    <fileset dir="${pluggable.ui.root}/CentraSiteControl/lib">
      <include name="*.jar"/>
    </fileset>
    <fileset dir="${pluggable.ui.root}/WEB-INF/lib">
      <include name="*.jar"/>
    </fileset>
  </path>


  <!-- default target, build all -->
  <target name="all" description="all" depends="jar, zip"/>

  <!-- establish jar file of plugin -->
  <target name="jar" depends="compile" description="jar">
    <mkdir dir="${lib}" />
    <jar destfile="${lib}/${plugin.name}.jar">
      <fileset dir="${classes}"/>
      <fileset dir="${src.dir}" includes="**/*.properties"/>
      <manifest>
        <section name="com/centrasite/control">
          <attribute name="Implementation-Title" value="${plugin.name}"/>
          <attribute name="Implementation-Version" value="1.0.0.0"/>
          <attribute name="Implementation-Vendor" value="${plugin.provider}"/>
        </section>
      </manifest>
    </jar>
  </target>

  <!-- compile java sources -->
  <target name="compile" description="compile" depends="">
    <mkdir dir="${plugin.dir}/accesspath" />
    <mkdir dir="${classes}" />
    <javac srcdir="${src.dir}" destdir="${classes}" debug="on"
           classpathref="plugin.class.path" />
  </target>

  <!-- Create plugin archive -->
  <target name="zip" description="package the plugin" depends="">
     <zip destfile="${plugin.name}.zip" basedir="${plugin.dir}/.."
                                      includes="${plugin.name}/lib/**,
                   ${plugin.name}/accesspath/**, ${plugin.name}/plugin.xml,
                                                 ${plugin.name}/*_SWT.xml,
                   ${plugin.name}/xml/** ${plugin.name}/*.html" />
  </target>


  <!-- Install plugin zip to PluggableUI -->
  <target name="install" description="Install plugin zip to PluggableUI">
      <java ↩
classname="com.softwareag.cis.plugin.ext.plugins.command.InstallPlugInCommand"
            fork="true">
```

```
            <arg value="-t" />
            <arg value="${pluggable.ui.root}" />
            <arg value="-z" />
            <arg value="${plugin.name}.zip" />
            <classpath>
                <fileset dir="${pluggable.ui.root}/WEB-INF/lib">
                    <include name="*.jar"/>
                </fileset>
            </classpath>
        </java>
    </target>

    <!-- Uninstall plugin zip from PluggableUI -->
    <target name="uninstall" description="Uninstall plugin zip from PluggableUI">
      <delete dir="${pluggable.ui.root}/${plugin.name}" />
    </target>

    <!-- Cleanup objects -->
    <target name="clean" description="clean classes lib and generated files">
      <delete dir="${classes}" />
      <delete dir="${plugin.dir}/accesspath" />
      <delete dir="${lib}" />
      <delete file="${plugin.name}.zip" />
    </target>

</project>
```

build.xml

The classpath for the build step must comprise all JAR files used by the UI. Add these JAR files to the build path of your java project also.

As mentioned above, in order to present a user-defined screen when the plug-in's **import** button is clicked, an XML file that describes the GUI must be located in the subdirectory *xml*. The example XML file simply prompts for a filename:

```
<?xml version="1.0" encoding="UTF-8"?>
<page model="com.importer.myfile.control.ImportMyFileAdapter">
    <pagebody>
        <rowarea withleftborder="false" withtopborder="false"
                 withrightborder="false" withbottomborder="false"
                 withtoppadding="false"
                 paddingleft="10" paddingright="10">
            <vdist height="30"></vdist>
            <itr>
                <label name="File:" width="100" asplaintext="true"></label>
                <fileupload2 width="100%" cfileprop="fileClientUrl"
                             fileprop="fileServerUrl"
                             method="fileLoaded"></fileupload2>
            </itr>
```

```
            <vdist height="30"></vdist>
        </rowarea>
    </pagebody>
    <statusbar></statusbar>
</page>
```

xml/ImportMyFile.xml

The Plug-In Environment

The master file of the plug-in is *plugin.xml*:

```
<plugin id="com.importer.myfile" order="101">
  <requiredPlugin id="com.centrasite.control" />
  <requiredPlugin id="com.softwareag.cis.plugin" />

  <!-- Import extension (point to command execution class) -->
  <extension point="com.centrasite.control.import"
             id="importMyFileCommand"
             class="com.importer.myfile.control.ImportMyFileCommand">
  </extension>
</plugin>
```

plugin.xml

This file specifies the command class, which is used to select the user's import function. It must be derived from `com.centrasite.control.extpt.AbstractImport`.

Note: This section does not explain all the details of the Java source file; its purpose is to indicate the code that must be modified to suit your environment.

```
package com.importer.myfile.control;

import com.centrasite.control.extpt.AbstractImport;
import com.centrasite.control.ActionContext;

public class ImportMyFileCommand extends AbstractImport
{
    static final String
           IMPORT_NAME     = "Import MyFile";  // Appears in the import list
    static final String
           HTML_PAGE       = "/ImportMyFile/ImportMyFile.html";
    static final String
           MY_IMAGE        = "../ImportMyFile/images/importMyFile.gif";
    static final String
           CALLING_ADAPTER = "com.importer.myfile.control.ImportMyFileAdapter";
```

```
                            // point to the adapter class

    public ImportMyFileCommand() {
    }

    public String getName() {
      return IMPORT_NAME;
    }

    public String getImageURL() {
      return MY_IMAGE;
    }

    public String getLayout() {
      return HTML_PAGE;
    }

    public String getPageDescription() {
      return "XPDL v.1 Importer";
    }

    public void execute(ActionContext actionContext) {
      actionContext.showPage(HTML_PAGE, getName(), CALLING_ADAPTER);
    }
}
```

src/com/importer/myfile/control/ImportMyFileCommand.java

This class defines the paths of the image file for your private icon, the HTML file used and the class of the import adapter.

Here is the frame of an import adapter:

```
package com.importer.myfile.control;

import java.util.Collection;
import javax.xml.registry.JAXRException;
import com.centrasite.control.AbstractBrowseCommand;
import com.centrasite.control.ActionContext;
import com.centrasite.control.Connector;
import com.centrasite.control.adapters.BaseAdapter;
import com.centrasite.control.adapters.util.ImportAdapter;
import com.centrasite.control.discovery.PromptYesNoHandler;
import com.centrasite.control.logged.action.LoggedExecutor;
import com.centrasite.control.logged.action.LoggedSchemaImport;
import com.centrasite.control.interfaces.Initializable;
import com.centrasite.jaxr.JAXRAccessor;
import com.softwareag.cis.plugin.interfaces.RunnableDeferred;

/**
```

```
 *  Import adapter
 */
public class ImportMyFileAdapter extends BaseAdapter
             implements Initializable,ImportAdapter
{
    private static final String TITLE = "Import MyFile";

    private String fileTmpUrl;
    private String fileAtServer;
    private String fileAtClient;

    public ImportMyFileAdapter() {
        fileAtServer = fileAtClient = fileTmpUrl = null;
    }

    public void initialize(Collection<Object> objs){}

    public void setOrganization(String org){}

    public boolean execute() {
        callFinish();
        return true;
    }

    public String getFileClientUrl() {
        return fileAtClient;
    }

    public void setFileClientUrl(String value) {
        fileAtClient = value;
    }

    public String getFileServerUrl() {
        return fileTmpUrl;
    }

    public void setFileServerUrl(String value) {
        fileTmpUrl = value;
    }

    public void fileLoaded() {
        fileAtServer = fileTmpUrl;
    }

    public void callCancel() {
        super.endProcess();
    }

    private static boolean isWhiteSpace(String s)
    {
        if (s == null || s.length() == 0) return true;
        for (int i=0 ; i < s.length() ; ++i) {
```

```
            if (!Character.isWhitespace(s.charAt(i))) return false;
        }
        return true;
    }

    /**
     * Called if "OK" button has been pressed
     */
    public void callFinish() {
        if (isWhiteSpace(fileAtServer))
        {
            outputMessage(MT_ERROR, "no file entered");
        }
        else
        {
           ImportMyFile ie = new ImportMyFile(getConnector(),
                                fileAtServer, fileAtClient);
           ie.doImport();
        }
    }

    /**
     * Class for importing specific files
     */
    private class ImportMyFile extends AbstractBrowseCommand
                               implements RunnableDeferred, PromptYesNoHandler
    {
        private Connector connector;
        private JAXRAccessor jaxr;

        public ImportMyFile(Connector connector, String fileAtServer,
                            String fileAtClient){
           // fileAtClient is the filename you can access
           this.connector = connector;
           this.jaxr = null;
        }


        public void doImport() {
           JAXRAccessor jaxr = null;
           try {
              LoggedSchemaImport lsi = new LoggedSchemaImport(getActionContext());
              jaxr = getJAXR();
              // write here your import code to registry and repository
              //setEventCallback( lsi.getEventCallback() );  // for event logging

              // your import code can use the methods
              // accept(javax.xml.registry.infomodel.RegistryObject ro)
              // warning(javax.xml.registry.infomodel.RegistryObject ro)
              // reject(javax.xml.registry.infomodel.RegistryObject ro)
              // of the LoggedEventCallback class, to log the status of your import
```

```
            new LoggedExecutor(lsi).execute();
         }
         catch (Exception e) {
            throw new RuntimeException(e);
         }
         finally {
            if (jaxr != null)
               jaxr.close();
         }
      }

      private JAXRAccessor getJAXR() throws JAXRException {
          if (jaxr == null)  {
              jaxr = new JAXRAccessor(connector.getRegistryUrl(),
                         connector.getUserName(), connector.getPassword());
          }
          return jaxr;
      }

      public void run() throws Exception {
          doImport();
      }

      public void handleYes(ActionContext actionContext) {
          doImport();
      }

      public void handleNo(ActionContext actionContext) {}

     public void executeCommand(ActionContext actionContext, String clientPath) {}

      public void executeCommand(ActionContext actionContext, String clientPath,
           String serverPath) {
          actionContext.executeDeferred(this);
      }

      public int getCategory() {
          return CATEGORY_MISC;
      }

      public String getImageURL() {
          return null;
      }

      public String getTitle() {
          return TITLE;
      }

      public String getName() {
          return "";
      }
```

```
        public String getLabel() {
            return "";
        }
    }
}
```

src/com/importer/myfile/control/ImportMyFileAdapter.java

Assuming that you have set up all the Java files correctly in the directory *src*, you should be able to build with the command:

```
ant -f build.xml jar all
```

This creates the plug-in-specific JAR file in the subdirectory *lib* and archives the necessary plugin files into the file *ImportMyFile.zip*.

Propagating the Plug-In

Having generated the plugin files, they must be propagated into the directory *PluggableUI* of the installed CentraSite Control. Thus, for example, the plug-in **"ImportMyFile"** (shown above) should have the following directory/file structure:

```
.../PluggableUI/ImportMyFile /
    accesspath /
        ImportMyFile.access
    images /
        ImportMyFile.gif
    lib /
        ImportMyFile.jar
    ImportMyFile.html
    ImportMyFile_JLIBS.html
    ImportMyFile_SWT.xml
    plugin.xml
```

You can do this by executing the following command, which installs ImportMyFile.zip into the PluggableUI folder:

```
ant -f build.xml install
```

Generating all additionally needed files

You only need this step if you are not able to install the necessary files directly in the webapps->PluggableUI direction. In this case, you have to build the structure shown above by yourself. You can automatically generate the .html files as well as the _SWT.xml and the .accesspath file using the Software AG Application Designer.

First edit the file stored in the following directory: *<TomcatWebAppsDir>*/PluggableUI/cis/config/cis-config.xml.

Insert the statement: plugindevelopment="true" in the following part:

```
designtimeclassloader="com.softwareag.cis.plugin.registry.loader.AdapterPluginClassLoader"
  enableadapterpreload="true"
  framebuffersize="3"
  plugindevelopment="true"
  loglevel=""
  logtoscreen="false"
  maxitemsinfieldcombo="100"
```

After that, restart the Software AG tomcat server. Now you can start the Application Designer with the following URL: *http://localhost:53307/PluggableUI/HTMLBasedGUI/workplace/ide.html*.

Navigate to "Tools & Documentation > Layout Manager" and choose the "ImportMyFile" Plugin as Application Project. It appears in the Layout Definitions List. Click on it and generate all additionally needed files using the button "Operations on multiple Items > (Re)Generate HTML pages".

Activating the Plug-In

To activate the plug-in

1 Restart the application server (Tomcat).

2 Start CentraSite Control.

3 Choose the Import function. You will see the name of your plug-in in the **Import as** selection list.

Select the name of your plug-in and click Next.

4 Enter the name of the file to be imported, then click Finish.

Deactivating the Plug-In

To deactivate the plug-in

1 Run the command:

```
ant -f build.xml uninstall
```

2 Restart the application server (Tomcat).

3 Start CentraSite Control.