

CentraSite Developer's Guide

Version 10.7

October 2020

This document applies to CentraSite 10.7 and to all subsequent releases.

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

Copyright © 2005-2020 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <https://softwareag.com/licenses/>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <https://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <https://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

Document ID: CS-DG-107-20201019

Table of Contents

About this Guide.....	5
Document Conventions.....	6
Online Information and Support.....	6
Data Protection.....	7
 1 API for JAXR.....	 9
Introduction to the CentraSite API for JAXR.....	10
Creating a JAXR-based Connection.....	10
Closing a JAXR-based Connection.....	11
Defining a Service.....	12
Service that Uses Another Service.....	12
Service with Additional Information.....	13
Pre-defined Classification Schemes (Taxonomies).....	13
CentraSite API for JAXR Reference Information.....	15
 2 Web Service Interfaces.....	 27
Introduction to the Web Service Interfaces.....	28
Approval Service.....	28
 3 Java Management Interface.....	 51
Introduction to Java Management Interface.....	52
Attributes and Operations.....	53
 4 Customizing CentraSite.....	 57
Customization of CentraSite Business UI.....	58
Customization of CentraSite Control.....	106
Implementation of Computed Attributes and Profiles.....	169
Customizing CentraSite i18n Messages.....	182
 5 Custom Policy Actions.....	 187
Custom Policy Actions.....	188
Computed Run-Time Actions.....	200
 6 Custom Reporting Searches.....	 209
Writing Your Own Reporting Search.....	210
Displaying List of Reporting Searches.....	219
Obtaining Details of Reporting Search.....	220
Copying Reporting Search.....	221
Sample XQueries for Reporting Searches.....	222
 7 Application Framework.....	 233

Introduction to Application Framework.....	234
Configuring Application Framework.....	236
Mapping Beans to Registry Objects with Annotations.....	238
Querying the Registry.....	247
Event Mechanism.....	256
Asset Types.....	257
Association Types.....	258
Lifecycle Management.....	259
Revision Management.....	261
Multi-User Scenarios.....	262
Setting the Classpath.....	263
Examples.....	263
 8 Importing Objects Using API.....	 265
Introduction to Importing Objects Using API.....	266
Invoking Importer from Java Program.....	266
Invoking Importer through Command Line.....	286
Invoking Importer Using SOAP API.....	291
Writing Your Own Importer.....	292
 9 Setting Up CentraSite Eclipse Plug-ins.....	 303
Installing CentraSite Plug-ins in Your Own Eclipse Environment.....	304
Connecting Eclipse to CentraSite.....	305
Using CentraSite Online Documentation.....	306

About this Guide

- Document Conventions 6
- Online Information and Support 6
- Data Protection 7

This guide describes how you can use programming interfaces to access and modify the CentraSite Registry Repository. Additionally, it describes how you can customize the CentraSite graphical user interfaces to suit the requirements or standards of your organization.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies service names and locations in the format <i>folder.subfolder.service</i> , APIs, Java classes, methods, properties.
<i>Italic</i>	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
Monospace font	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the symbol.
[]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at <http://documentation.softwareag.com>.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at <https://empower.softwareag.com/>.

You can find product information on the Software AG Empower Product Support website at <https://empower.softwareag.com>.

To submit feature/enhancement requests, get information about product availability, and download products, go to [Products](#).

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the [Knowledge Center](#).

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at https://empower.softwareag.com/public_directory.aspx and give us a call.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at <http://techcommunity.softwareag.com>. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

1 API for JAXR

■ Introduction to the CentraSite API for JAXR	10
■ Creating a JAXR-based Connection	10
■ Closing a JAXR-based Connection	11
■ Defining a Service	12
■ Service that Uses Another Service	12
■ Service with Additional Information	13
■ Pre-defined Classification Schemes (Taxonomies)	13
■ CentraSite API for JAXR Reference Information	15

Introduction to the CentraSite API for JAXR

The CentraSite API for JAXR (Java Application Program Interface for eXtensible Markup Language Repositories) is based on the Java API for XML Registries (JAXR) standard. CentraSite supports JAXR capability level 1. In addition, it has some extensions that enable you to exploit specific functions of CentraSite. The reader should be an experienced Java programmer, with knowledge of XML and the concepts of enterprise repositories.

CentraSite extends the JAXR standard with the following:

- Ability to create user-defined object types.

CentraSite extends the JAXR object model by user-defined types, which may have triggers and operations attached. Correspondingly, the CentraSite JAXR-based extensions interface extends the JAXR query interface and allows you to search user-defined objects.

- Ability to use XQuery to access to the stored data.

CentraSite allows a client to access the stored data directly using XQuery through the XQJ-based (XQuery API for Java) interface.

Creating a JAXR-based Connection

➤ To create a JAXR-based connection

1. Ensure that the CLASSPATH includes directories that contain all JAR files in the CentraSite `redist` folder.

Note:

If you have activated an e-mail policy, the CLASSPATH must additionally include the file `mail.jar`, which you can find in the `rts/bin` folder.

2. Do one of the following:

- Start your client program with the parameter:

```
-Djavax.xml.registry.ConnectionFactoryClass=com.centrasite.jaxr.  
ConnectionFactoryImpl
```

- Set this property during program startup:

```
System.setProperty("javax.xml.registry.ConnectionFactoryClass",  
    "com.centrasite.jaxr.ConnectionFactoryImpl");
```

3. Create a factory by running the command `connFactory`, along with the appropriate code snippet:

```
ConnectionFactory connFactory = ConnectionFactory.newInstance();
```

4. Supply the `queryManagerURL` to the connection.

```
Properties p = new Properties();
```

```
p.setProperty("javax.xml.registry.queryManagerURL",
    "http://localhost:53307/CentraSite/CentraSite");
```

Note:

- In CentraSite, the `lifeCycleManagerURL` is always the same as the `queryManagerURL`, hence it need not be specified.
- The port number, in the example above specified as 53307, may need to be changed to suit your local configuration.

5. Set the `BrowserBehaviour` option.

```
p.setProperty("com.centrasite.jaxr.BrowserBehaviour", "yes");
connFactory.setProperties(p);
```

Enabling `BrowserBehaviour` mode is the preferred way of creating a JAXR-based connection. This is beneficial for several reasons. The `BrowserBehaviour` mode uses a less strict locking pattern, and this can result in an increased number of parallel read and update operations. For example in CentraSite Control, while one user is looking at some asset, another user can update the same asset in parallel. In the same scenario without `BrowserBehaviour`, the update would fail as the necessary lock cannot not be granted.

Moreover, with `BrowserBehaviour` mode, the assets cached on the client side are refreshed often. After an asset is read, it is refreshed in the cache if it is returned as the result of a subsequent query with a newer timestamp.

6. Create the connection and set the user credentials. The `setCredentials()` method expects a `Set` containing a `java.net.PasswordAuthentication` object.

```
Connection connection = connFactory.createConnection();

HashSet credentials = new HashSet(1);
credentials.add(new PasswordAuthentication("userid",

                                         "password".toCharArray()));

connection.setCredentials(credentials);
```

7. Construct the other objects once the connection is created.

```
RegistryService regService = connection.getRegistryService();
BusinessLifeCycleManager lcManager =
    regService.getBusinessLifeCycleManager();
BusinessQueryManager bqManager = regService.getBusinessQueryManager();
```

Closing a JAXR-based Connection

A JAXR-based connection uses some resources in the CentraSite XML Server.

Software AG recommends that you ensure the connection is closed after the client program has been processed successfully or failed due to a JAXR-based client failure. Otherwise the resources are released only after a non-activity timeout, which might hinder parallel users.

> To close a JAXR-based connection

- Call the method:

```
connection.close()
```

Defining a Service

A service is provided by an organization. It should have a name and a description, and the details are specified by service bindings, which are further detailed by specification links. The following code snippet, which assumes that the providing organization is known, shows how to create a new service:

```
Organization providingOrganization = ...;

Service service = m_lcManager.createService("service name");
service.setProvidingOrganization(providingOrganization);
InternationalString description =
    lcManager.createInternationalString("service description");
service.setDescription(description);

ServiceBinding serviceBinding = ...;
// create service binding with specification links

service.addServiceBinding(serviceBinding);

ArrayList serviceList = new ArrayList();
serviceList.add(service);
lcManager.saveServices(serviceList);
// save service and related modified objects
```

Service that Uses Another Service

If a service calls another service, this should be modeled with the pre-defined Uses association.

```
Service callingService = ...;
Service calledService = ...;

// find the "Uses" concept
ClassificationScheme associationType = bqManager.findClassificationSchemeByName(
    Collections.singleton(FindQualifier.EXACT_NAME_MATCH), "AssociationType");
Concept usesConcept =
    bqManager.findConceptByPath("/") + associationType.getKey().getId() + "/Uses");

// create association of type "Uses"
Association usesAssociation =
    lcManager.createAssociation(calledService, usesConcept);

// callingService is now the source object of the association
callingService.addAssociation(usesAssociation);

ArrayList associationList = new ArrayList();
associationList.add(usesAssociation);

// save association and related modified objects
```

```
lcManager.saveAssociations(associationList, false);
```

Service with Additional Information

Each JAXR-based object instance may be supplied with arbitrary additional information. JAXR uses the slot mechanism to provide this kind of extensibility.

Note:

JAXR allows arbitrary strings as slot names. The CentraSite implementation stores a slot by creating an XML element whose tag name is the slot name. Consequently, a slot name should be a valid XML QName. If a QName has a non-null URI, the lexical representation of the slotname is the URI enclosed in curly braces, followed by the local-name, for example {myUri}mySlotname.

The following code snippet shows how to add a slot to a service object:

```
Service service = ...;

Slot slot = lcManager.createSlot("{myUri}mySlotName", "slotValue", null);
service.addSlot(slot);

ArrayList serviceList = new ArrayList();
serviceList.add(service);
lcManager.saveServices(serviceList);
```

Pre-defined Classification Schemes (Taxonomies)

The CentraSite registry has several pre-defined classification schemes:

- All the classification schemes that are defined in the JAXR standard.
- A classification scheme for the products using CentraSite. Thus, each registry object can be classified with its product. This makes it easy to find all registry objects originating from a particular product.

The name of this classification scheme is `Products` and its member concepts are `CentraSite` and `products` that use `CentraSite`.

- A classification scheme for database management systems: This can be used to classify data sources by the type of the database management system they represent.

The name of this classification scheme is `Databases` and its member concepts are:

- `Adabas`
- `Tamino`
- `DB2`
- `Enabler`
- `MSSQL`
- `Oracle`

- A classification scheme for content types: This can be used to classify external links with their content type/MIME type.

The name of this classification scheme is `Content Type`.

- A classification scheme for the types of objects in the CentraSite repository: This can be used to classify external links with their repository object type.

The name of this classification scheme is `RepositoryObjectType` and its member concepts are:

- BPEL
- BPELObject
- CustomComponent
- Documentation
- DTD
- E-mailEvent
- Emerger
- FileEvent
- HTML
- Icon
- JAR
- JMSEvent
- Layout
- Ontology
- Payload
- ProjectFolder
- ReportDefinition
- ScheduledTask
- Sequence
- SOAP
- Template
- TypeIcon
- WSDD
- WSDL

- XML
- XSD
- XSLT
- Some external classification schemes used for UDDI mapping:
 - ClassificationGroup
 - Object
 - UseType
 - uddi-org:protocol:http
 - uddi-org:protocol:soap
 - uddi-org:wSDL:address
 - uddi-org:wSDL:categorization:protocol
 - uddi-org:wSDL:categorization:transport
 - uddi-org:wSDL:portTypeReference
 - uddi-org:wSDL:types
 - uddi-org:xml:localName
 - uddi-org:xml:namespace

CentraSite API for JAXR Reference Information

This section explains the differences between the JAXR standard and our APIs, particularly, the CentraSite-specific extensions to the JAXR standard.

Creating User-Defined Objects

In addition to the pre-defined object types such as organizations, services, and associations, CentraSite allows you to define your own object types. Once such a type has been created using the CentraSite Control, a corresponding concept exists in the `ObjectType` classification scheme.

➤ To create an instance of a user-defined object type

1. Create a `RegistryEntry` object.
2. Classify it with the type concept.

The following code example assumes that a user-defined type `{User-Uri}UserType` exists:

```
RegistryEntry userTypeObject
    = (RegistryEntry)lcManager.createObject(LifeCycleManager.REGISTRY_ENTRY);
```

```
// find the "{User-Uri}UserType" concept
ClassificationScheme objectType
    = bqManager.findClassificationSchemeByName(null, "ObjectType");
Concept userTypeConcept
    = bqManager.findConceptByPath("/") + objectType.getKey().getId()
        + "{User-Uri}UserType");

// create classification
Classification userTypeClassification
    = lcManager.createClassification(userTypeConcept);
userTypeObject.addClassification(userTypeClassification);

/*
 * from now on the userTypeObject is of type "UserType", and
 * userTypeObject.getObjectType() will return a concept equal to
 * userTypeConcept
 */

// save object
ArrayList objectList = new ArrayList();
objectList.add(userTypeObject);
lcManager.saveObjects(objectList);
```

Direct XQuery Access to the Stored Data

A CentraSite JAXR client can call XQJ (XQuery API for Java technology) functionality directly in order to access the registry data. JAXR also uses XQJ to access the registry.

The CentraSite `CentraSiteConnection` maintains an `XQConnection` object, which it uses for its own purposes as well as for direct client access. The client can get this object as follows, assuming JAXR-based connection is already present:

```
Connection jaxrCon = ...;
XQConnection xqjCon = ((CentraSiteConnection)jaxrCon).getXQConnection();
```

As both the client and JAXR use the same XQJ connection, the following restrictions apply (assuming the client uses JAXR and XQJ in parallel):

- The client must not call any JAXR-based `save...` method if there is an open transaction, because JAXR performs the `save...` methods as one atomic operation based on an XQJ transaction.
- The client should never close the XQJ connection; instead, close the JAXR-based connection. This action cleans up anything else.

Unique Keys

This implementation does not support client supplied keys. The method `RegistryObject.setKey()` displays an `UnsupportedCapabilityException`. CentraSite rejects client-supplied keys.

Simultaneous Database Access and Locking

The CentraSite implementation stores all RegistryObjects in a common repository, which is a database. If multiple JAXR-based clients (or, to be more precise, multiple JAXR-based connections) are active simultaneously, it is possible that they might read and update the data in the common database concurrently.

Multiple clients that update a RegistryObject must be synchronized in order to prevent update losses. Usually, this is handled by the underlying database's locking mechanism. However, since it is likely that many JAXR-based clients would be browsing or searching the repository and only a few JAXR-based clients would be modifying data, the CentraSite implementation has been optimized to allow maximum concurrent access. In particular, if one or more JAXR-based clients are reading a RegistryObject, another JAXR-based client may update it concurrently.

For example, if a user has opened CentraSite Control to look for a particular object and then keeps the UI open for a protracted period (maybe even for several days), this should not prevent other users from updating that object.

Locks for read access are therefore relatively permissive, but of course it must be ensured that two JAXR-based clients cannot modify the same object at the same time. This is achieved as follows:

When a JAXR-based client starts to modify a RegistryObject, JAXR acquires an exclusive lock for this object from the database management system. This prevents any other client from updating the same object at the same time. When the JAXR-based client saves the modified object, the lock is released as a side-effect of calling `LifeCycleManager.saveObjects()`. Alternatively, if the JAXR-based client decides to discard the changes, it should release the lock by calling `CentraSiteConnection.rollback()`.

With this locking behavior, there are two principal scenarios when two JAXR-based clients attempt to modify the same object at the same time. In order to modify an object, the JAXR-based client always has to read it first, modify the Java instance, and then call `saveObjects()` to write the modified object back to the database.

Scenario A

JAXR-based Client A

1. Read a RegistryObject.
3. Start to modify the object. This automatically locks the object.

JAXR-based Client B

2. Read the same RegistryObject.
4. Start to modify the object. The attempt to lock the object fails and a `LockNotAvailableException` occurs.

As long as client A holds the exclusive lock for the object, client B is unable to modify it.

Scenario B

JAXR-based Client A

1. Read a RegistryObject.
3. Start to modify the object. This automatically locks the object.
4. Save the object. This releases the lock.

JAXR-based Client B

2. Read the same RegistryObject.
5. Start to modify the object. The attempt to lock the object fails and an `ObjectOutdatedException` occurs.

In scenario B, client A has finished making its changes and has released the lock, so the lock is now available for acquisition by another client, for example client B. However, client B's local copy of the object does not reflect the current database status of the object, which has been modified in the meantime by client A. If client B were allowed to save the object, client A's modifications would be overwritten.

To avoid this, each RegistryObject has a last-modification date. When a lock is acquired, the API checks whether the last-modification date of the object in the database is the same as the last-modification date of the client's local copy of the object. If the dates are not the same, an `ObjectOutdatedException` occurs. This ensures that updates are not lost and that all modifications are based on the latest state of the object.

Immediately before the `ObjectOutdatedException` occurs, the API cleans up its internal structures. When the client encounters the exception, it should release all references to the RegistryObject and then re-read it. This should return the latest copy of the object from the database; the client can now continue to make the necessary modifications to this clean copy.

Caller

The caller identifies himself by issuing `Connection.setCredentials()`. The corresponding user object is retrieved from the registry using the name given in the credentials. If the user record does not yet exist, CentraSite returns an error message.

Note:

Each user must be registered in the CentraSite registry before being used as the caller.

Here, the user name is the name attribute as inherited from the RegistryObject interface. It should not be confused with the user's `PersonName`.

The caller must be known before a connection can be used. In other words, `setCredentials()` is required, otherwise a security error occurs.

Note:

The user name must be unique in the CentraSite registry.

Semantics of Remove Operations

There are several methods that allow an object to be removed from its parent. Depending on the kind of object, the remove operation has different effects.

- **Associations, Classifications, External Identifiers, Service Bindings, Specification Links.** If such an object is removed from its parent and the parent is then saved, the object is automatically deleted as it cannot exist as a standalone object. Remove these objects using the following methods:
 - `RegistryObject.removeClassification()`
 - `RegistryObject.setClassifications()`
 - `RegistryObject.removeAssociation()`
 - `RegistryObject.setAssociations()`
 - `RegistryObject.removeExternalIdentifier()`
 - `RegistryObject.setExternalIdentifiers()`
 - `ServiceBinding.removeSpecificationLink()`
 - `Service.removeServiceBinding()`
- **Other Objects.** Other objects are de-linked from their parents during the remove operation. They continue to exist as separate objects. If the parent object is saved, the removed objects are also automatically saved.

The remove operations for these objects are:

- `ClassificationScheme.removeChildConcept()`
- `Concept.removeChildConcept()`
- `Organization.removeUser()`
- `Organization.removeService()`
- `Organization.removeChildOrganization()`
- `RegistryObject.removeExternalLink()`
- `RegistryObject.setExternalLinks()`
- `RegistryPackage.removeRegistryObject()`

Delete Operation

Deleting an object means deleting it from the persistent store. Optionally, the delete operation can be called with an `objectType` parameter, which is one of the pre-defined `LifeCycleManager` interface names. If this parameter is specified, only objects of that type are accepted for delete. The interface names shown in the following list are allowed for a deletion; all others are rejected with an `InvalidRequestException`.

- `LifeCycleManager.ASSOCIATION`
- `LifeCycleManager.CLASSIFICATION`
- `LifeCycleManager.CLASSIFICATION_SCHEME`
- `LifeCycleManager.CONCEPT`
- `LifeCycleManager.EXTERNAL_IDENTIFIER`
- `LifeCycleManager.EXTERNAL_LINK`
- `LifeCycleManager.ORGANIZATION`
- `LifeCycleManager.REGISTRY_ENTRY`
- `LifeCycleManager.REGISTRY_PACKAGE`
- `LifeCycleManager.SERVICE`
- `LifeCycleManager.SERVICE_BINDING`
- `LifeCycleManager.SPECIFICATION_LINK`
- `LifeCycleManager.USER`

Objects have relationships to each other: some relationships prohibit object deletion, while other relationships are automatically cleaned up during deletion.

RegistryObject

In general, an attempt to delete a registry object is rejected if:

- it is a new object, that is, it has not yet been saved, or
- it is the target of an association.

Deleting a registry object has the following side-effects:

- Removes the object from all its packages; update the packages.
- De-links the object from all its external links; update the external links.
- Deletes all associations whose source object is the object to be deleted.
- Deletes all classifications whose classified object is the object to be deleted.

- Deletes all external identifiers whose registry object is the object to be deleted.

Association

1. Remove the association from its source object.
2. Update the source object. This automatically deletes the association.

AuditableEvent

It is not possible to delete an auditable event explicitly.

Classification

1. Remove the classification from its classified object.
2. Update the classified object. This automatically deletes the classification.

ClassificationScheme

1. Reject deletion if there are child concepts; else
2. Delete the classification scheme.

Concept

1. Reject deletion if there are child concepts; else,
2. Remove the concept from its parent object.
3. Update the parent object.
4. Delete the concept.

ExternalIdentifier

1. Remove the external identifier from its registry object.
2. Update the registry object. This automatically deletes the external identifier.

ExternalLink

1. Reject deletion if there are linked objects; else,
2. Delete the external link.

Organization

1. Reject deletion if there are child organizations, services, or users; else,

2. Remove the organization from its parent organization.
3. Update the parent organization.
4. Delete the organization.

RegistryEntry

1. Delete the registry entry.

RegistryPackage

1. Reject deletion if there are member objects; else,
2. Delete the registry package.

Service

1. Remove the service from its organization.
2. Update the organization.
3. Delete all service bindings whose service is the service to be deleted.
4. Delete the service.

ServiceBinding

1. Remove the service binding from its service.
2. Delete all specification links whose service binding is the service binding to be deleted.
3. Update the service. This automatically deletes the service binding.

SpecificationLink

1. Remove the specification link from its service binding.
2. Update the service binding's enclosing service. This automatically deletes the specification link.

User

1. Remove the user from its organization.
2. Update the organization.
3. Delete the user.

Unsupported Methods

The following methods are not supported and an `UnsupportedCapabilityException` occurs:

- `RegistryService.getDeclarativeQueryManager()`
- `RegistryService.makeRegistrySpecificRequest()`

Unsupported FindQualifiers

The following `FindQualifiers` are not supported:

- `COMBINE_CLASSIFICATIONS`
- `SERVICE_SUBSET`
- `SOUNDEX`

Using Wildcards

The wildcard character, which is the percent (%) character, represents zero or more characters. Thus, for example, the search string `ABC%DEF` finds all strings that begin with `ABC` and end with `DEF`, with any number of characters in between. The search string `ABC%DEF%` finds all strings that begin with `ABC` and include `DEF` anywhere else. If you do not include a wildcard character in the search string, the search assumes that there is a wildcard character at the end of the search string, unless the find qualifier `EXACT_NAME_MATCH` is specified. Thus, for example, if you specify `ABC` as the search string, the search in fact looks for and finds strings that match the pattern `ABC%`, that is, all strings that begin with the characters `ABC`.

Using Namespaces

Some names, for example type names and slot names, comprise a namespace and a name. When programming a JAXR-based client, these names must be represented in the following format:

```
{namespace}name
```

In other words, the namespace is enclosed in curly braces and is used as a prefix for the name.

Strings in this format are used in the following methods:

```
for objectType in CentraSiteQueryManager.findObjects()
for typeName in CentraSiteQueryManager.getTypeDescription()
for name in LifecycleManager.createSlot()
for slotName in ExtensibleObject.getSlot()
```

Method createSlot

The method `createSlot` in the interface `LifeCycleManager` takes 3 parameters; its signatures are as follows:

```
Slot createSlot (String name, String value, String slotType)
Slot createSlot (String name, Collection values, String slotType)
```

The `CentraSite` implementation accepts any value of type `String`, or a null reference, for the third parameter, `slotType`. This parameter is stored with the slot, but it is not interpreted in any way. Note, however, that the JAXR standard does not indicate how this parameter should be interpreted; it might, for example, be interpreted as indicating the data type of the slot in some future implementation. Software AG recommends specifying the `slotType` as an `xs:string`.

Caching Considerations

This section describes various aspects of caching behavior as it affects the API.

JAXR-based Caching Strategy

Objects that are retrieved from the registry by means of the `CentraSite` API for JAXR are stored in a cache by the JAXR-based connection. All objects stored in the cache are inspected from time to time by the Java garbage collector, which may delete them if there are no references to them from the application.

Any object reference that results from a call to `getRegistryObject()`, `getRegistryObjects()` or any of the `find` methods is, if possible, resolved from the cache. If an application already holds a reference to an object that resulted from any of these calls, the reference is present in the cache, and the call returns the same Java reference.

There are situations, however, where the cache is cleared completely. This occurs, for example, after executing `saveObjects` or `deleteObjects`. Any Java reference that is retrieved after the cache is cleared is different from a reference that was retrieved before the cache is cleared.

Note:

This does not affect data integrity, since objects read cannot be concurrently updated.

Caching in User Interfaces

The `CentraSite` user interfaces, that is, `Control`, `Business UI`, and `Eclipse`, browse JAXR-based data. This means that they make use of the JAXR-based caching mechanism, but they do not block concurrent updates. `Control`, `Business UI`, and `Eclipse` users should be aware that, in general, the data display does not immediately reflect changes that another user may make.

Note:

This does not affect data integrity in the sense that outdated data may be the source of any updates.

You can see the current data at any time by clicking the **Refresh** button.

Dynamically Loaded JAR Files

CentraSite locally caches dynamically-loaded JAR files. You should be aware that the date and time of the cached files are compared with the date and time of the library files whenever a new connection is created, the JAR files in the cache are refreshed if they are found to be out of date. This could mean that processing continues with a newer version of a JAR file after a connection has been created.

Note:

Problems may arise if a custom security manager has been implemented, because the connection to the database is refused.

Cache Location

CentraSite uses the following strategy to determine the location of the cache store:

- If the system property `com.softwareag.centrasite.dynloader.cache-dir` is defined, then its value is used as the location of the cache store.
- Otherwise, the location of the cache store is derived from:
 1. A directory whose name is taken from the system property `java.io.tmpdir`.
 2. A sub-directory whose name is constructed from the string `CentraSite`, a package name, and the string `Jars`.

2 Web Service Interfaces

■ Introduction to the Web Service Interfaces	28
■ Approval Service	28

Introduction to the Web Service Interfaces

This chapter describes some of the web services that CentraSite provides. You can obtain a complete list of services at:

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

Where *server* is the machine on which the Software AG Runtime is running and *port* is the port on which Software AG Runtime is listening (use port 53307 if CentraSite is configured to use the default Software AG Runtime port number). For example:

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

The information in this chapter is intended for developers who want to integrate custom applications or third-party tools with CentraSite using web services.

CentraSite provides the following web services for each of the predefined importers:

- ImportWsdService
- ImportXsdService
- ImportXPDLService
- ApprovalService

As an example, this chapter describes the ApprovalService in detail.

Approval Service

The Approval service provides a set of operations that enables you to programmatically interact with CentraSite's approval system. Using the Approval service, you can develop client applications that let users view requests that they have submitted for approval and let approvers accept or reject these requests.

The WSDL for the Approval service is located here:

```
http://server:port/wsstack/services/ApprovalService?wsdl
```

Where *server* is the machine on which the Software AG Runtime is running and *port* is the port on which Software AG Runtime is listening (port 53307 if CentraSite is configured to use the default Software AG Runtime port number).

The following lists the operations that the Approval service provides:

- [getPendingApprovals](#)
- [getApprovalRequests](#)
- [getApprovalActions](#)
- [approve](#)
- [reject](#)

- [getApprovalHistory](#)
- [revertPendingStateChange](#)

Invoking Operations from the Approval Service

You can perform the invoking operations from the approval service by specifying the authenticated user or by specifying the location of the approval log.

Specifying the Authenticated User

The Approval service returns results that are specific to the *authenticated user* (that is, the user who invokes the operation). For example, when a client application invokes the `getPendingApprovals` operation, the operation returns the set of approval requests that require the *authenticated user's* approval.

The authenticated user is identified by the basic http authentication credentials that the client application provides when it invokes an operation in the Approval service. The supplied credentials must identify an active user account on the instance of CentraSite to which the client application is connecting. If the client application submits invalid credentials, the Approval service returns a SOAP fault.

Specifying the Location of the Approval Log

All of the operations provided by the Approval service have an input parameter called `locationCentraSite`. This parameter identifies the address of the CentraSite registry or repository whose approval log is to be queried. A client application must specify the `locationCentraSite` parameter if the registry or repository is running anywhere other than its default location (that is, port 53307 on the machine where the Approval service is running).

If the registry or repository is running at its default location, a client does not have to specify the `locationCentraSite` parameter.

Retrieving the List of Approval Requests that a User Has Submitted

You can use the [getApprovalRequests](#) operation to retrieve the list of approval requests that the authenticated user has submitted to CentraSite. By default, this operation returns *all* the approval requests that a user has submitted. However, you can optionally set the `objectType`, `submittedAfter`, `submittedBefore`, and `status` parameters to filter the list by the following criteria:

- The type of object on which the request was submitted
- The time period during which the request was submitted
- The status of the request (for example, retrieve only those requests that have not yet been approved)

You would use this operation, for example, to show users a list of their requests that are pending approval.

Tip:

This operation provides functionality like that of the **Approval Requests** list in CentraSite Control.

ApprovalRequestList Message

The [getApprovalRequests](#) operation (and other operations provided by the Approval service) returns an [ApprovalRequestList](#) message. This message contains an array of `ApprovalRequest` elements. Each `ApprovalRequest` element in the array represents a single approval request and contains the following information:

- The key of the approval request object (this key is required to perform operations that act directly on a specific approval request)
- The key of the user who submitted the approval request
- The date on which the approval request was submitted
- The key of registry object for which the approval request was submitted
- The type of object for which the approval request was submitted
- The status of the request
- Remarks, if any, that were submitted with the approval request

The approval requests in the array are not sorted.

The `ApprovalRequestList` message also returns an attribute called `count` that indicates the total number of approval requests in the result set. If the operation does not find any approval requests that satisfied the operation's criteria, then no element is returned in `ApprovalRequest[]` and the `count` value is zero.

If you want to receive the result, set a few entries at a time instead of all at once, you can use the `scroll` parameter in the request message to specify which block of entries you want the operation to return. For more information about using the `scroll` parameter, see [Scrolling Through the List of Returned Approval Requests](#).

Getting Details about the Actions of the Approvers Associated with a Request

Once you have an [ApprovalRequestList](#), you can use the [getApprovalActions](#) operation to obtain detailed information about the approvers associated with any request in the list.

The `getApprovalsActions` operation takes an approval request key as input (which you can get from the [ApprovalRequestList](#)) and returns the set of approvers associated with the specified request. (You can specify multiple keys if you want to get the details for multiple approval requests.)

The `getApprovalsActions` operation returns an [ApprovalActionResult](#) message. The `ApprovalAction[]` array in this message identifies the set of approvers associated with a particular approval request. Each `ApprovalAction` element in this array contains the following information:

- The key for the approver (that is, the key to the `User` object that represents the approver)
- The status of the approver's action on this request is as follows:
 - If the request has not yet been processed to completion (that is, it has not yet been approved or rejected) and the approver has not taken any action on the request, then the status is `Pending`.
 - If the approver has approved the request, then the status is `Approved`.
 - If the approver has rejected the request, then the status is `Rejected`.
 - If the request has been processed to completion (that is, it has been approved or rejected), approvers who did not make the approval decision have the status as `No Action`. (If the approval request was auto-approved, all of the approvers have the status as `No Action`.)

Tip:

This operation provides functionality like that of the **Approval Requests** list when you use CentraSite Control to display the details for an approval request.

Approving or Rejecting Approval Requests

➤ To enable a user to approve or reject a request

1. Use the [“getPendingApprovals” on page 33](#) operation to obtain the list of requests that require the user's approval.
2. Apply the [“approve” on page 37](#) or [“reject” on page 37](#) operation to the requests in the list according to the approval decisions that the user makes.

When you invoke the approve or reject operation, you must specify the key of the approval request on which the operation is to act. You can obtain this key from the [ApprovalRequestList](#) message that was returned by the [getPendingApprovals](#) operation.

Note:

You can apply the approve or reject operation to a single approval request or to multiple requests.

The approve and reject operations return an [ApprovalRequestList](#) message. This message contains the approval requests that were approved or rejected by the operation.

Tip:

This operation provides functionality like that of the **Pending Approvals** list in CentraSite Control.

Scrolling Through the List of Returned Approval Requests

The [getPendingApprovals](#), [getApprovalRequests](#), and [getApprovalHistory](#) operations each return an array of approval requests (that is, their result set) in an [ApprovalRequestList](#) message. In certain cases, the result set can be quite large (for example, if you were to retrieve the entire Approval History log). Instead of receiving the entire result set in a single message, you can use the `scroll` parameter to retrieve the results in blocks of a specified size (for example, 15 entries at a time). For example, you might use this feature to display when an approval requests a page at a time in your client application.

To receive a specified block of results, set the following elements in the `scroll` parameter when you invoke the [getPendingApprovals](#), [getApprovalRequests](#), and [getApprovalHistory](#) operation.

In this element...	Specify...
start	The first element in the block that you want to retrieve (where 1 represents the first element in the entire set of results).
number	The total number of elements that you want to retrieve in that block (that is, the size of the block).

For example, let's say you are using the [getApprovalHistory](#) operation, and you want to retrieve the contents of the log 20 entries at a time. To do this you would:

Invoke...	Set...	Set...
getApprovalHistory	<code>scroll.start = 1</code>	<code>scroll.number = 20</code>
getApprovalHistory again	<code>scroll.start = 21</code>	<code>scroll.number = 20</code>
getApprovalHistory again	<code>scroll.start = 41</code>	<code>scroll.number = 20</code>

You would continue until you reach the end of the result set.

To determine that you have reached the end of the result set, you can check the value in the `count` parameter in the [ApprovalRequestList](#). This parameter reports the total number of entries in the entire result set.

Note:

If the last block in the set contains fewer entries than what you specify in `scroll.number`, the operation returns the remaining entries in that last block. If the element that you specify in `scroll.start` does not exist in the result set, the operation returns an empty list.

Reverting a Pending Approval Request

There might be times when you need to retract a pending request from the approval system. For example, if a request that is awaiting approval requires the approval of a user who has left the company, the request can be stuck in pending mode. To resolve this condition, you must remove that request from the approval system and resubmit it (after updating the approver group).

When you have an approval request that is stuck in the pending mode, you can use the [revertPendingStateChange](#) operation to remove the request from the approval system. This operation also reverts the object that was pending approval to its previous state so that a user can submit the object for approval again.

Note:

When you invoke this operation, you must specify the key of the registry object whose state you want to revert. You can obtain this key from the approval request that is stuck in pending mode. (You would need to retrieve that request and the object's key, using one of the operations that returns an [ApprovalRequestList](#).)

This operation returns a `revertPendingStateChangeResponse` message. The value of the `revertedState` parameter in this message reports the lifecycle state of the object on which the `revertPendingStateChange` operation was executed. For example, if you execute this operation on an object whose lifecycle state is pending a change from state A to state B, the operation reverts the object to state A and returns state A in the `revertedState` parameter.

Note:

Only users in the CentraSite Administrator role are permitted to execute the `revertPendingStateChange` operation. If the authenticated user is not a member of this role, the operation returns a SOAP fault.

Approval Service Operations

This section describes the various operations that the Approval service provides.

getPendingApprovals

This operation returns a list of the approval requests that are awaiting the authenticated user's approval (where the authenticated user is the user who invoked the `getPendingApprovals` service). You can optionally filter the list by object type and submission date.

Input Message

Parameter Name	Description
<code>locationCentraSite</code>	<p>(Optional). (String). The address of the CentraSite registry or repository from which you want to retrieve the approval requests. The registry or repository runs at the following URL:</p> <p><code>http://server:port/CentraSite/CentraSite</code></p> <p>Where <i>server</i> is the machine on which the CentraSite registry or repository is running and <i>port</i> is the port on which Apache is configured to listen for requests (port 53307 if CentraSite is configured to use the default Apache port number).</p> <p>If you do not specify <code>locationCentraSite</code>, the Approval service uses the following default URL:</p>

Parameter Name	Description				
	http://localhost:53307/CentraSite/CentraSite				
objectType	(Optional). (String). (Array). If you want to retrieve approval requests for only certain object types, use this element to specify the types by name. Note: You must specify the type's schema name, not its display name. You can find the schema name on the type's Asset Type Details page in CentraSite Control.				
submittedAfter	(Optional). (DateTime). If you want to retrieve requests after a particular date, specify that date in this element.				
submittedBefore	(Optional). (DateTime). If you want to retrieve requests before a particular date, specify that date in this element.				
locale	(Optional). (String). The locale in which you want the results to be returned.				
scroll	(Optional). (Scroll). If you want to return a particular block of entries from the result set, specify the following values in the <code>scroll</code> element. <table><tr><td>start</td><td>(Integer). The first entry that you want to include in the block (where 1 represents the first entry in the entire result set).</td></tr><tr><td>number</td><td>(Optional). (Integer). The number of entries to be returned in the block of approval requests. If you specify a <code>start</code> value, but no <code>number</code> value, then the remainder of the result set is returned.</td></tr></table>	start	(Integer). The first entry that you want to include in the block (where 1 represents the first entry in the entire result set).	number	(Optional). (Integer). The number of entries to be returned in the block of approval requests. If you specify a <code>start</code> value, but no <code>number</code> value, then the remainder of the result set is returned.
start	(Integer). The first entry that you want to include in the block (where 1 represents the first entry in the entire result set).				
number	(Optional). (Integer). The number of entries to be returned in the block of approval requests. If you specify a <code>start</code> value, but no <code>number</code> value, then the remainder of the result set is returned.				

Output Message

For more information, see [ApprovalRequestList](#).

getApprovalRequests

This operation returns the list of requests that the authenticated user has submitted for approval (where the authenticated user is the user who invoked the `getApprovalRequests` service). You can optionally filter the list by object type, submission date, and approval status.

Input Message

Parameter Name	Description										
locationCentraSite	<p>(Optional). (String). The address of the CentraSite registry or repository from which you want to retrieve the approval requests. The registry or repository runs at the following URL:</p> <p><code>http://server:port/CentraSite/CentraSite</code></p> <p>Where <i>server</i> is the machine on which the CentraSite registry or repository is running and <i>port</i> is the port on which Apache is configured to listen for requests (port 53307 if CentraSite is configured to use the default Apache port number).</p> <p>If you do not specify <code>locationCentraSite</code>, the Approval service uses the following default URL:</p> <p><code>http://localhost:53307/CentraSite/CentraSite</code></p>										
status	<p>(Optional). (String). If you want to retrieve only requests with a specified approval status, specify one of the following values:</p> <table> <tr> <th>Specify...</th><th>To retrieve...</th></tr> <tr> <td>In Progress</td><td>Approval requests that are pending (awaiting approval).</td></tr> <tr> <td>Approved</td><td>Approval request that have been approved (excluding requests that were auto-approved).</td></tr> <tr> <td>Rejected</td><td>Approval requests that have been rejected.</td></tr> <tr> <td>No Action</td><td>Approval requests that were auto-approved.</td></tr> </table>	Specify...	To retrieve...	In Progress	Approval requests that are pending (awaiting approval).	Approved	Approval request that have been approved (excluding requests that were auto-approved).	Rejected	Approval requests that have been rejected.	No Action	Approval requests that were auto-approved.
Specify...	To retrieve...										
In Progress	Approval requests that are pending (awaiting approval).										
Approved	Approval request that have been approved (excluding requests that were auto-approved).										
Rejected	Approval requests that have been rejected.										
No Action	Approval requests that were auto-approved.										
objectType	<p>(Optional). (String). (Array). If you want to retrieve approval requests for only certain object types, specify the types by name in this element.</p> <div> <p>Note: You must specify the type's schema name, not its display name. You can find the schema name on the type's Asset Type Details page in CentraSite Control.</p> </div>										
submittedAfter	(Optional). (DateTime). If you want to retrieve requests after a particular date, specify that date in this element.										
submittedBefore	(Optional). (DateTime). If you want to retrieve requests before a particular date, specify that date in this element.										
locale	(Optional). (String). The locale in which you want the results returned.										
scroll	(Optional). (Scroll). If you want to return a particular block of entries from the result set, specify the following values in the <code>scroll</code> element:										

Parameter Name	Description
start	(Integer). The first entry that you want to include in the returned block of approval requests (where 1 represents the first entry in the entire result set).
number	(Optional). (Integer). The number of entries to be returned in the block of approval requests. If you specify a start value, but no number value, then the remainder of the result set is returned.

Output Message

For more information, see [ApprovalRequestList](#).

getApprovalActions

This operation returns detailed information about specified approval requests.

Input Message

Parameter Name	Description
locationCentraSite	(Optional). (String). The address of the CentraSite registry or repository from which you want to retrieve the approval requests. The registry or repository runs at the following URL: <code>http://server:port/CentraSite/CentraSite</code> Where <i>server</i> is the machine on which the CentraSite registry or repository is running and <i>port</i> is the port on which Apache is configured to listen for requests (port 53307 if CentraSite is configured to use the default Apache port number). If you do not specify <code>locationCentraSite</code> , the Approval service uses the following default URL: <code>http://localhost:53307/CentraSite/CentraSite</code>
approvalRequestKeys	(String). (Array). The keys for the approval requests whose details you want to retrieve.
locale	(Optional). (String). The locale in which you want the results returned.

Output Message

For more information, see [ApprovalActionResult](#).

approve

This operation approves specified approval requests.

Input Message

Parameter Name	Description
<code>locationCentraSite</code>	<p>(Optional). (String). The address of the CentraSite registry or repository on which the approval requests reside. The registry or repository runs at the following URL:</p> <p><code>http://server:port/CentraSite/CentraSite</code></p> <p>Where <i>server</i> is the machine on which the CentraSite registry or repository is running and <i>port</i> is the port on which Apache is configured to listen for requests (port 53307 if CentraSite is configured to use the default Apache port number).</p> <p>If you do not specify <code>locationCentraSite</code>, the Approval service uses the following default URL:</p> <p><code>http://localhost:53307/CentraSite/CentraSite</code></p>
<code>approvalRequestKeys</code>	(String). (Array). The keys for the requests that are to be approved.
<code>comment</code>	(Optional). (String). A comment from the approver.
<code>locale</code>	(Optional). (String). The locale in which you want the results returned.

Output Message

For more information, see [ApprovalRequestList](#) (contains the requests that were approved).

reject

This operation rejects the specified approval requests.

Input Message

Parameter Name	Description
<code>locationCentraSite</code>	<p>(Optional). (String). The address of the CentraSite registry or repository on which the approval requests reside. The registry or repository runs at the following URL:</p> <p><code>http://server:port/CentraSite/CentraSite</code></p> <p>Where <i>server</i> is the machine on which the CentraSite registry or repository is running and <i>port</i> is the port on which Apache is</p>

Parameter Name	Description
	<p>configured to listen for requests (port 53307 if CentraSite is configured to use the default Apache port number).</p> <p>If you do not specify <code>locationCentraSite</code>, the Approval service uses the following default URL:</p> <p><code>http://localhost:53307/CentraSite/CentraSite</code></p>
<code>approvalRequestKeys</code>	(String). (Array). The keys for the requests that are to be rejected.
<code>comment</code>	(Optional). (String). A comment from the approver.
<code>locale</code>	(Optional). (String). The locale in which you want the requests returned.

Output Message

For more information, see [ApprovalRequestList](#) (contains the requests that were rejected).

getApprovalHistory

This operation returns entries from the approval history log based on specified search criteria. If the user belongs to the CentraSite Administrator role, he or she receives all entries in the log. If the user belongs to the Organization Administrator role, he or she receives all log entries for his or her organization. Otherwise, the user receives only those approval requests that he or she has submitted.

Input Message

Parameter Name	Description
<code>locationCentraSite</code>	<p>(Optional). (String). The address of the CentraSite registry or repository on which the approval history log resides. The registry or repository runs at the following URL:</p> <p><code>http://server:port/CentraSite/CentraSite</code></p> <p>Where <i>server</i> is the machine on which the CentraSite registry or repository is running and <i>port</i> is the port on which Apache is configured to listen for requests (port 53307 if CentraSite is configured to use the default Apache port number).</p> <p>If you do not specify <code>locationCentraSite</code>, the Approval service uses the following default URL:</p> <p><code>http://localhost:53307/CentraSite/CentraSite</code></p>
<code>status</code>	(Optional). (String). If you want to retrieve only requests with a specified approval status, specify one of the values shown here:

Parameter Name	Description	
	Specify...	To retrieve...
	In Progress	Approval requests that are pending (that is, awaiting approval).
	Approved	Approval requests that have been approved (excluding requests that were auto-approved).
	Rejected	Approval request that have been rejected.
	No Action	Approval requests that were auto-approved.
objectType	(Optional). (String). (Array). If you want to retrieve approval requests for only certain object types, specify the types by name in this element. <div> Note: You must specify the type's schema name, not its display name. You can find the schema name on the type's Asset Type Details page in CentraSite Control. </div>	
submittedAfter	(Optional). (DateTime). If you want to retrieve requests after a particular date, specify that date in this element.	
submittedBefore	(Optional). (DateTime). If you want to retrieve requests before a particular date, specify that date in this element.	
locale	(Optional). (String). The locale in which you want the results returned.	
scroll	(Optional). (Scroll). If you want to return a specified block of entries from the result set, specify the following values in the <code>scroll</code> element.	
	start	(Integer). The first entry that you want to include in the block (where 1 represents the first entry in the entire result set).
	number	(Optional). (Integer). The number of entries to be returned in the block. If you specify a <code>start</code> value, but no <code>number</code> value, the remainder of the result set is returned.

Output Message

For more information, see [ApprovalRequestList](#).

revertPendingStateChange

This operation removes an object that is pending approval from the approval system, and returns the object to its prior lifecycle state. Only users that belong to the CentraSite Administrator role can execute this operation.

Input Message

Parameter Name	Description
locationCentraSite	<p>(Optional). (String). The address of the CentraSite registry or repository on which the object resides. The registry or repository runs at the following URL:</p> <p><code>http://server:port/CentraSite/CentraSite</code></p> <p>Where <i>server</i> is the machine on which the CentraSite registry or repository is running and <i>port</i> is the port on which Apache is configured to listen for requests (port 53307 if CentraSite is configured to use the default Apache port number).</p> <p>If you do not specify <code>locationCentraSite</code>, the Approval service uses the following default URL:</p> <p><code>http://localhost:53307/CentraSite/CentraSite</code></p>
key	(String). The key of the object whose state you want to revert.

Output Message

`revertPendingStateChangeResponse`

Parameter Name	Description
revertedState	(String). The lifecycle state to which the object was reverted by the revertPendingStateChange operation. For example, if you executed this operation on an object whose state was pending a change from state A to state B, the operation would return state A in the <code>revertedState</code> parameter.

ApprovalRequestList

This data structure holds a list of approval requests.

Parameter Name	Description				
ApprovalRequest[]	<p>An array of <code>ApprovalRequest</code> elements. Each <code>ApprovalRequest</code> entry in the array represents one approval request and has the following structure:</p> <table><tr><td>name</td><td>(Optional). (String). The name of the approval request (as specified by the Approval Flow Name parameter in the approval policy action).</td></tr><tr><td>requestor</td><td>(String). The key that identifies the user who submitted the approval request.</td></tr></table>	name	(Optional). (String). The name of the approval request (as specified by the Approval Flow Name parameter in the approval policy action).	requestor	(String). The key that identifies the user who submitted the approval request.
name	(Optional). (String). The name of the approval request (as specified by the Approval Flow Name parameter in the approval policy action).				
requestor	(String). The key that identifies the user who submitted the approval request.				

Parameter Name	Description											
	registryObject	(String). The key of the registry object on which the user is requesting approval.										
	requestType	(String). The type of event that triggered the approval request (for example, Pre-State Change).										
	reasonForRequest	(Optional). (String). The remark (if any) that was assigned to the request by the approval policy action.										
	key	(String). The approval request's key.										
	status	(String). The state of the approval request. The value of this element will be one of the following:										
		<table><tr><th>Value</th><th>Description</th></tr><tr><td>In Progress</td><td>The approval request is pending (that is, awaiting approval).</td></tr><tr><td>Approved</td><td>The approval request has been approved.</td></tr><tr><td>Rejected</td><td>The approval request has been rejected.</td></tr><tr><td>No Action</td><td>The approval request was auto-approved.</td></tr></table>	Value	Description	In Progress	The approval request is pending (that is, awaiting approval).	Approved	The approval request has been approved.	Rejected	The approval request has been rejected.	No Action	The approval request was auto-approved.
Value	Description											
In Progress	The approval request is pending (that is, awaiting approval).											
Approved	The approval request has been approved.											
Rejected	The approval request has been rejected.											
No Action	The approval request was auto-approved.											
	submittedDate	(DateTime). The date on which the request was submitted for approval.										
scroll		(Optional). (String). The scroll values, if any, that were submitted when the operation that produced this ApprovalRequestList was invoked.										
	start	(Integer). The start value that was specified in the input message when the operation was invoked.										
	number	(Integer). (Optional). The number value that was specified in the input message when the operation was invoked.										
count		(Number). The number of approval requests in the entire result set.										

ApprovalActionResult

This data structure holds the details for a specified set of approval requests.

Parameter Name and Description

ApprovalActionList[]

An array of ApprovalActionList elements. Each ApprovalActionList entry in the array holds the details for one approval request and has the following structure:

ApprovalAction[]	(String). The key to the approval request.
ApprovalRequestKey	An array of ApprovalAction elements. Each ApprovalAction element in the array holds the approval details for one approver. This array contains one entry for each approver in the approver group. Each ApprovalAction element has the following structure:

approver	(String). The key that identifies the user who is the approver.
----------	---

status comment	(Optional). (String). A remark from the approver (typically indicating why he or she approved or rejected the request).
----------------	---

(String). The approver's decision on the request. Possible values are shown below:

- Pending
The approver has not taken action on the request.
- Approved
The approver approved the request.
- Rejected
The approver rejected the request.
- No Action
The request has been approved or rejected, however, this approver did not make the approval decision on the request. This can also indicate that the request was auto-approved.

Search Service Operations

The Search service provides a set of operations that supplies various information about registered services inside CentraSite.

The WSDL for the Search service is located here:

<http://server:port/wsstack/services/SearchService?wsdl>

where *server* is the machine in which the Software AG Runtime is running and *port* is the port on which the Software AG Runtime is listening (port 53307 if CentraSite is configured to use the default Software AG Runtime port number).

The following lists the operations that the Search service provides:

- `findAllServices`
- `findServices`
- `findOrganizations`
- `getAllServiceDetails`
- `getServiceDetails`
- `getAllAssociatedServices`
- `getAssociatedServices`

findServices and findAllServices

The `findServices` operation returns a list of services in CentraSite. You can optionally filter the list by specific criteria.

The `findAllServices` operation, in addition, returns a list of registered services in Integration Server.

Input Message

Parameter Name	Description
<code>locationCentraSite</code>	<p>(Optional). (String). The address of the CentraSite registry repository (CRR) from which you want to retrieve the search requests. The registry or repository runs at the following URL:</p> <p><code>http://server:port/CentraSite/CentraSite</code></p> <p>Where <i>server</i> is the machine on which the CRR is running and <i>port</i> is the port on which CentraSite is listening to requests.</p> <p>If you do not specify <code>locationCentraSite</code>, the Search service uses the following default URL:</p> <p><code>http://localhost:53307/CentraSite/CentraSite</code></p>
<code>keyword</code>	<p>(Mandatory). (String). The pattern for matching an exact service name or %, or * for matching all services.</p>
<code>organizationKey</code>	<p>(Optional). (String). (Array). The UDDI key or the organization name where the services belong to.</p>

Parameter Name	Description
owner	(Optional). (String). (Array). The first and last name of the owner of services to be selected. The first and last name must be separated with a ,.
lifeCycleStateName	(Optional). (String). The name of a valid lifecycle state currently active with the service type. Only the services matching that lifecycle state will be selected. You may specify multiple lifecycle state name elements to match more states.
product	(Optional). (String). A product name the service is classified with. If specified it must be a valid name among the Products taxonomy of CentraSite, for example, ARIS. You may specify multiple product name elements to match more product classifications.
locale	(Optional). (String). The locale in which you want the results to be returned.
sortCriteria	Sort criteria for returning the result in a sorted order. Possible values are: <ul style="list-style-type: none">■ SortByNameAsc■ SortByNameDesc■ SortByDateAsc■ SortByDateDesc■ Organization
scroll	(Optional). (Scroll). If you want to return a particular block of entries from the result set, specify the following values in the scroll element.
	start (Integer). The first entry that you want to include in the block (where 1 represents the first entry in the entire result set).
	number (Optional). (Integer). The number of entries to be returned in the block of approval requests. If you specify a start value, but no number value, then the remainder of the result set is returned.

Output Message

ServiceInfo[]

Parameter Name	Description
ServiceInfo - An array of ServiceInfo elements. Each ServiceInfo entry represents one registered service. The ServiceInfo message has the following details:	
key	The UDDI key of the service.
name	The display name of the service.
wsdlURL	The URL of the service's WSDL in the CentraSite repository.
lifeCycleState	The name of the current lifecycle state
owner	The user who created the service.
organizationKey	The UDDI key of the organization which owns the service.
organizationName	The organization which owns the service.
version	The user-assigned version identifier for the service.

Example SOAP Request

Returns the first 10 services for a particular organization, sorted by name in ascending order:

```
<soapenv:Body>
  <findAllServices>
    <locationCentraSite></locationCentraSite>
    <keyword>*</keyword>
    <organizationKey>uddi:207ff1cc-25c5-544c-415c-5d98ea91060c
      </organizationKey>

    <owner></owner>
    <product></product>
    <locale>EN</locale>
    <sortCriteria>SortByNameAsc</sortCriteria>
    <scroll>
      <start>1</start>
      <number>10</number>
    </scroll>
  </findAllServices>
</soapenv:Body>
```

findOrganizations

The findOrganizations operation returns a list of organizations in CentraSite. You can optionally filter the list by specific criteria.

Input Message

Parameter Name	Description
locationCentraSite	(Optional). (String). The address of the CentraSite registry repository (CRR) from which you want to retrieve the search requests. The registry or repository runs at the following URL:

Parameter Name	Description				
	<p><code>http://server:port/CentraSite/CentraSite</code></p> <p>Where <i>server</i> is the machine on which the CRR is running and <i>port</i> is the port on which CentraSite is listening to requests.</p> <p>If you do not specify <code>locationCentraSite</code>, the Search service uses the following default URL:</p> <p><code>http://localhost:53307/CentraSite/CentraSite</code></p>				
<code>keyword</code>	(Mandatory). (String). The pattern for matching an exact organization name or %, or * for matching all organizations.				
<code>locale</code>	(Optional). (String). The locale in which you want the results to be returned.				
<code>sortCriteria</code>	<p>Sort criteria for returning the result in a sorted order. Possible values are:</p> <ul style="list-style-type: none">■ <code>SortByNameAsc</code>■ <code>SortByNameDesc</code>■ <code>SortByDateAsc</code>■ <code>SortByDateDesc</code>■ <code>Organization</code>				
<code>scroll</code>	<p>(Optional). (Scroll). If you want to return a particular block of entries from the result set, specify the following values in the <code>scroll</code> element.</p> <table><tr><td><code>start</code></td><td>(Integer). The first entry that you want to include in the block (where 1 represents the first entry in the entire result set).</td></tr><tr><td><code>number</code></td><td><p>(Optional). (Integer). The number of entries to be returned in the block of approval requests.</p><p>If you specify a <code>start</code> value, but no <code>number</code> value, then the remainder of the result set is returned.</p></td></tr></table>	<code>start</code>	(Integer). The first entry that you want to include in the block (where 1 represents the first entry in the entire result set).	<code>number</code>	<p>(Optional). (Integer). The number of entries to be returned in the block of approval requests.</p> <p>If you specify a <code>start</code> value, but no <code>number</code> value, then the remainder of the result set is returned.</p>
<code>start</code>	(Integer). The first entry that you want to include in the block (where 1 represents the first entry in the entire result set).				
<code>number</code>	<p>(Optional). (Integer). The number of entries to be returned in the block of approval requests.</p> <p>If you specify a <code>start</code> value, but no <code>number</code> value, then the remainder of the result set is returned.</p>				

Output Message

OrganizationInfo[]

Parameter Name	Description
<code>OrganizationInfo</code>	An array of OrganizationInfo elements. Each OrganizationInfo entry represents one organization. The OrganizationInfo message has the following details:

Parameter Name	Description
key	The UDDI key of the organization.
name	The display name of the organization.
description	The description for the organization.

Example SOAP Request

Returns information for the organization *myorg*:

```
<soapenv:Body>
  <findOrganizations>
    <locationCentraSite>
      http://localhost:53305/CentraSite/CentraSite
    </locationCentraSite>
    <keyword>myorg</keyword>
    <locale>DE</locale>
    <sortCriteria></sortCriteria>
  </findOrganizations>
</soapenv:Body>
```

getServiceDetails and getAllServiceDetails

The `getServiceDetails` operation returns a detailed information about a particular service in CentraSite. You can optionally filter the list of service information by specific criteria.

The `getAllServiceDetails` operation, in addition, returns a list of additional properties for the service.

Input Message

Parameter Name	Description
locationCentraSite	<p>(Optional). (String). The address of the CentraSite registry repository (CRR) from which you want to retrieve the search requests. The registry or repository runs at the following URL:</p> <p><code>http://server:port/CentraSite/CentraSite</code></p> <p>Where <i>server</i> is the machine on which the CRR is running and <i>port</i> is the port on which CentraSite is listening to requests.</p> <p>If you do not specify <code>locationCentraSite</code>, the Search service uses the following default URL:</p> <p><code>http://localhost:53307/CentraSite/CentraSite</code></p>
serviceKey	(Optional). (String). (Array). The UDDI key of the service.
locale	(Optional). (String). The locale in which you want the results to be returned.

Output Message

ServiceDetail[]

Parameter Name	Description
ServiceDetail - An array of ServiceDetail elements for each of the service. The ServiceDetail message has the following details:	
key	The UDDI key of the service.
name	The display name of the service.
wsdlURL	The URL of the service's WSDL in the CentraSite repository.
lifeCycleState	The name of the current lifecycle state
owner	The user who created the service.
organizationKey	The UDDI key of the organization which owns the service.
organizationName	The organization which owns the service.
version	The user-assigned version identifier for the service.
property	service namespace - The target namespace of the service.
OperationInfo[] - An array of OperationInfo for the service.	
key	The UDDI key of the service.
name	The display name of the service.
description	The description for the service.
property	■ input - The input message. ■ output - The output message.

Example SOAP Request

Returns detailed information for two particular services:

```
<soapenv:Body>
  <getServiceDetails>
    <locationCentraSite></locationCentraSite>
    <serviceKey>uddi:372ad925-d207-11e4-8844-ad99cc9211e8</serviceKey>
    <serviceKey>uddi:65f11014-c8b8-11e4-9dd8-c736d339af26</serviceKey>
    <locale></locale>
  </getServiceDetails>
</soapenv:Body>
```


getAssociatedServices and getAllAssociatedServices

The `getAssociatedServices` operation returns information about services that are associated to a particular service in CentraSite. You can optionally filter the list of service information by specific criteria.

Input Message

Parameter Name	Description
<code>locationCentraSite</code>	<p>(Optional). (String). The address of the CentraSite registry repository (CRR) from which you want to retrieve the search requests. The registry or repository runs at the following URL:</p> <p><code>http://server:port/CentraSite/CentraSite</code></p> <p>Where <i>server</i> is the machine on which the CRR is running and <i>port</i> is the port on which CentraSite is listening to requests.</p> <p>If you do not specify <code>locationCentraSite</code>, the Search service uses the following default URL:</p> <p><code>http://localhost:53307/CentraSite/CentraSite</code></p>
<code>assetKey</code>	(Optional). (String). (Array). The UDDI key of the service.
<code>associationType</code>	(Optional). (String). The name of the association type, for example, Uses, through which the asset references a service.

Output Message

`ServiceDetail[]`

Parameter Name	Description
<p><code>ServiceDetail</code> - An array of <code>ServiceDetail</code> elements for each of the service. The <code>ServiceDetail</code> message has the following details:</p>	
<code>key</code>	The UDDI key of the service.
<code>name</code>	The display name of the service.
<code>wsdlURL</code>	The URL of the service's WSDL in the CentraSite repository.
<code>lifeCycleState</code>	The name of the current lifecycle state
<code>owner</code>	The user who created the service.
<code>organizationKey</code>	The UDDI key of the organization which owns the service.
<code>organizationName</code>	The organization which owns the service.
<code>version</code>	The user-assigned version identifier for the service.

Parameter Name	Description
property	service namespace - The target namespace of the service.
OperationInfo[] - An array of OperationInfo for the service.	
key	The UDDI key of the service.
name	The display name of the service.
description	The description for the service.
property	<ul style="list-style-type: none">■ input - The input message.■ output - The output message.

3 Java Management Interface

■ Introduction to Java Management Interface	52
■ Attributes and Operations	53

Introduction to Java Management Interface

Use the CentraSite Java Management Interface to manage the CentraSite Registry or Repository. With the CentraSite Java Management Interface, you can:

- Monitor certain parameters of the CentraSite Registry or Repository.
- Change certain parameters of the CentraSite Registry or Repository. CentraSite parameters are known here as attributes.
- Perform operations such as starting and stopping the CentraSite Registry or Repository.

The JMX-based CentraSite Java management Interface is provided as an open MBean (managed bean) that interfaces to CentraSite.

To activate the CentraSite Java Management Interface, the MBean must be registered in an MBeanServer, which must run on the same host as the CentraSite Registry or Repository.

The CentraSite Java management interface is based on the Java Management Extensions (JMX) standard and the Java Management Extensions so that it can be used with JMX MBeanServers that are based on this standard.

Recent version of Java contains a JMX MBeanServer that can be requested by the ManagementFactory class.

If no other MBeanServer is running, you can have a look at the CentraSite Java Management Interface by registering the MBean by the MBeanServer of a Java process. Add the following three lines to the Java code:

```
MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();
ObjectName csAdmin = new ObjectName("CentraSite:id=CentraSiteAdminImp");
mbs.createMBean("com.centrasite.jmx.admin.CentraSiteAdminImpl", csAdmin);
```

For correct operation of the MBean, the following points must be fulfilled:

- The name of the MBean must be an ObjectName. In our case, it is CentraSite:id=CentraSiteAdminImp

- The CLASSPATH must include the following JAR files:

```
<CentraSiteInstall_Directory>/rts/jmx/CentraSiteJMXAdmin.jar
<CentraSiteInstall_Directory>/rts/jmx/CentraSiteJMXAdmin-L10N.jar
<CentraSiteInstall_Directory>/rts/jmx/CentraSiteAdminAPI.jar
<CentraSiteInstall_Directory>/rts/bin/inmUtil.jar
<CentraSiteInstall_Directory>/rts/bin/inmUtilConf.jar
<CentraSiteInstall_Directory>/rts/bin/log4j.jar
```

- The PATH must include the <CentraSiteInstall_Directory>/bin directory.
- Other system properties may be required, depending on the environment. For example, to use JConsole, the Java process must be started with the following Java system property:

```
-Dcom.sun.management.jmxremote
```

Note:

The above example shows how the CentraSite Java Management Interface works for the default MBean server of a JVM. In a production environment the integration in a MBean server may be different.

The CentraSite Java Management Interface works with Java version 5.0.

Attributes and Operations

The following sections describe the attributes and operations used in the Java Management Interface.

Attributes

Each attribute corresponds to a CentraSite parameter. The name, type, access mode (R = read-only; RW = read and write), current value, and description of each of the following attributes are output, for example, through the CentraSite Java Management console.

cache size

The size of the cache, in megabytes. If you change this attribute, the CentraSite Registry or Repository is automatically restarted to activate the new value.

max threads

The maximum number of threads used. If you change this attribute, the CentraSite Registry or Repository is automatically restarted to activate the new value.

max users

The maximum number of users that can be active concurrently. If you change this attribute, the CentraSite Registry or Repository is automatically restarted to activate the new value.

non-activity timeout

The session timeout period, in seconds. If no activity has occurred in a session for this period of time, the changes are rolled back and the session is terminated. If you change this attribute, the change takes effect immediately.

state

The current state of the CentraSite Registry or Repository. You cannot change this value.

transaction timeout

The maximum transaction duration, in seconds. If you change this attribute, the change takes effect immediately.

Registry or Repository Start and Stop Operations

The CentraSite Java Management Interface provides access to the following operations. For each operation that has one or more parameters, the name, type, and description of each parameter are output and the value of each parameter can be input. If you access the CentraSite Java Management Interface using console software (for example, a web browser), the operation is initiated when you select **Invoke**.

- Start the CentraSite Registry or Repository
- Stop the CentraSite Registry or Repository in normal mode
- Stop the CentraSite Registry or Repository in the specified mode

Possible termination modes are:

Code	Meaning
0 (normal)	Terminates the server session normally and waits for the current active processing to finish. The maximum waiting time (in seconds) can be set with the attribute <code>transaction timeout</code> .
1 (rollback)	Terminates the server immediately. User transactions that have not finished are rolled back.
2 (abort)	Terminates the server session immediately. The processing is immediately stopped. Crash dump files are written. Using this option initiates an automatic repair (autorepair) the next time the server is started.

- Back up the contents of the CentraSite Registry or Repository. Write the backup file to the default location.
- Back up the contents of the CentraSite Registry or Repository. Write the backup file to the specified location.

Back up the CentraSite Registry or Repository, writing the backup file to the default location or to the specified location. In either case, the backup identification is returned. Also, the backup is listed in the output of the `list all backups` operation. A backup is done to freeze the current state of the CentraSite Registry or Repository.

- Restore the contents of the CentraSite Registry or Repository from the latest backup with or without recovery.
- Restore the contents of the CentraSite Registry or Repository from the specified backup with or without recovery.

The `restore` operation can only be used when the CentraSite Registry or Repository is not active. It is used to restore the CentraSite Registry or Repository to the state that was stored in a previous backup. If you want to restore the most recent backup, you do not have to specify the identification of the backup.

Repository changes that occur between one backup and the next are stored in session logs. When restoring from a backup, you can optionally include (with recovery) or omit (without recovery) the session log data.

- Delete the specified backup file.

A backup that is no longer required can be deleted. Deleting a backup removes the backup spaces but the associated session log data is not removed, since it may be needed if the database has to be recovered. The backup file to be deleted is specified by means of the backup identification.

- List all backups.

Creates a list of CentraSite Registry or Repository backups. Each entry contains the corresponding backup identifier.

- Show more information about the last operation.

Shows additional information about the most recently processed command. Additional information can be displayed for the following operations: start, stop, back-up, restore, delete backup. If an operation fails, this command can be used to find the reason for the failure.

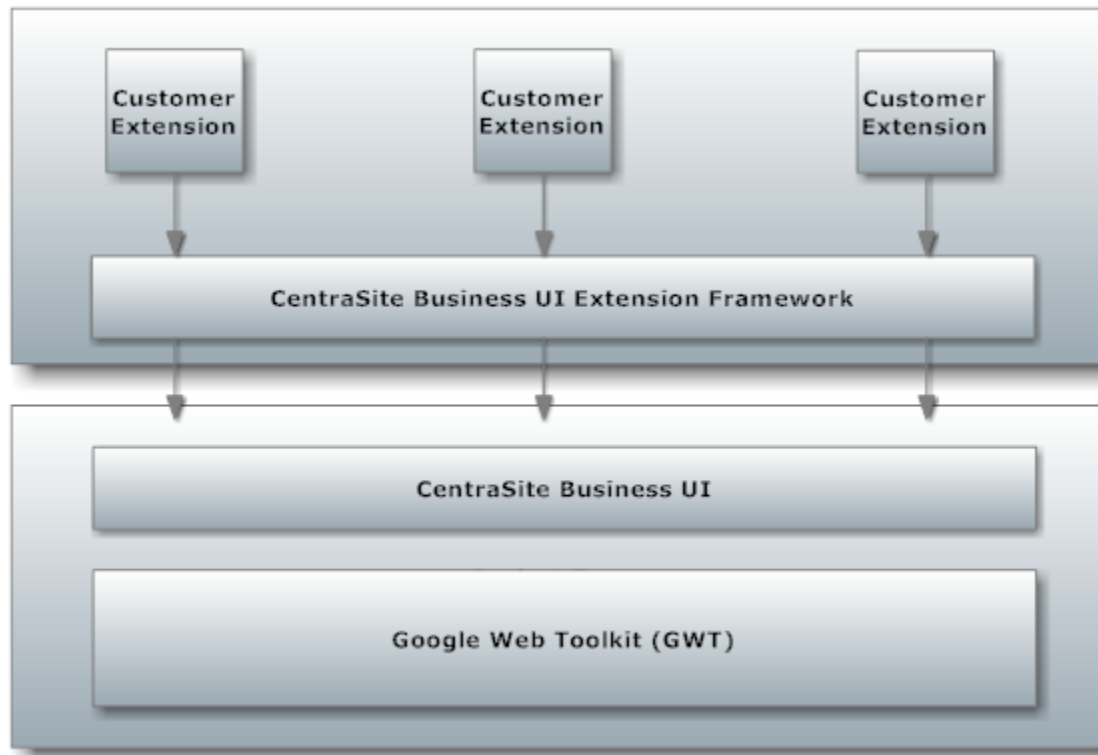
4 Customizing CentraSite

■ Customization of CentraSite Business UI	58
■ Customization of CentraSite Control	106
■ Implementation of Computed Attributes and Profiles	169
■ Customizing CentraSite i18n Messages	182

Customization of CentraSite Business UI

This section describes the CentraSite Business UI's pluggable architecture. You can extend CentraSite Business UI's functionality by adding your own features with appropriate widgets and JavaScripts. The CentraSite Business UI extension framework is a plug-in to a core infrastructure, in other words, the core infrastructure provides extension points where CentraSite Business UI extension framework is plugged in. The core infrastructure comprises of the CentraSite Business UI built on top of the Google Web Toolkit (GWT), which provides the widget classes for CentraSite.

The following figure illustrates the pluggable architecture:



Extensions are implemented as widgets. The points in the code at which extensions can be added are called extension points. CentraSite Business UI offers extension points that allow you to implement or extend the following features:

- Provide an alternative login screen.
- Provide an alternative screen for requesting an account.
- Extend the search dialog by additional search conditions.
- Replace the standard asset details view.
- Extend the set of available actions in the asset's details page.

CentraSite Business UI Configuration Files

When you start Software AG Runtime, property settings in the **centrasite.xml** files determine the features of the CentraSite Business UI

Configuration File Location	Description
<code><CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\system\conf</code>	Contains property settings that are installed for the CentraSite Business UI. Throughout this document, we use the term <i>system configuration file</i> when referring to this standard system file.
	Important: Do not modify the contents of this file unless asked to do so by Software AG.
<code><CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\custom\conf</code>	Contains properties that modify the installed settings in the system configuration file. Throughout this document, we use the term <i>custom configuration file</i> when referring to this customizable file.
	Modify this file, if you want to modify the property settings for the CentraSite Business UI.

Customizing CentraSite Login Page

The Login page used to access CentraSite Business UI can be customized to suit your requirements. You can change the logos used, colors, text, time formats, and layouts. You can also define links that navigate you to the Welcome page or guest access page, the user registration, and Take a Tour link of the CentraSite Business UI.

The standard Login page contains a header section, a login section, and a footer section. The following figure depicts the schematic layout of the Login page:

The diagram illustrates the layout of the CentraSite login page, organized into three main sections:

- Header:** Contains the "Blue Flag Company Logo".
- Main Content:** Features a central login form with the following elements:
 - "Login Product Logo"
 - "Username" input field
 - "Password" input field
 - A checkbox labeled "Remember Me"
 - "Log In" button
 - Links for "Login As Guest" and "Request An Account"
 - "Take A Tour Logo" and "Take A Tour" buttons
- Footer:** Includes "CopyRight" on the left and "Powered By Logo" on the right.

The header section displays the company logo. You can change the company logo and colors used to display the logo or hide the logo as part of the page customization. You can change the product name and the individual field labels and change the font and color used to display them. You can optionally choose to hide the product name as part of the page customization.

The footer section shows the copyright information. You can hide the copyright information as part of the page customization.

➤ To customize and install the customized login page

1. Stop Software AG Runtime.
2. Customize the Login page configuration.

The standard configuration file **centrasite.xml** delivered with the CentraSite kit contains all of the names of the CentraSite Business UI extension points, including the extension point for the Login page configuration. You must create a copy of the extension point for the Login page configurations in the customization file, **centrasite.xml**, in order to define a customized Login page. For more information, see [“CentraSite Login Page Configuration” on page 61](#).

You can find the customization file, **centrasite.xml**, on `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\custom\conf`.

3. Start Software AG Runtime.

The changes incorporated are now visible in the Login page.

CentraSite Login Page Configuration

The element `<GUIConfiguration>` in the **centrasite.xml** file contains the properties for customization of CentraSite Business UI. The Login page configuration includes configuring the header, content panel, and the footer.

Note:

Make sure that you specify an image file using its relative path in the `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\custom\images` directory. In a Windows environment, use forward slashes instead of backward slashes in the path name.

Header Configuration

Layout Component	Configuration Property	Description
CENTRASITE BY SOFTWARE AG	BlueFlagCompanyLogo	<p>Contains property settings that are used to modify the company logo in the Login page of CentraSite Business UI.</p> <p>To modify the company logo, configure the property statement:</p> <pre><BlueFlagCompanyLogo tooltip="CS_MSG_INMBX_LBL_PAGE_TITLE"> images/custom/ centrasite_by_sag_stacked.png </BlueFlagCompanyLogo></pre>

Content Panel Configuration

Layout Component	Configuration Property	Description
Welcome Please Log in	LoginProductLogo	<p>Contains property settings that are used to modify the CentraSite login information in the Login page of CentraSite Business UI.</p> <p>To modify the CentraSite login information, configure the property statement:</p> <pre><LoginProductLogo tooltip="CS_MSG_INMBX_LBL_PAGE_TITLE"> images/custom/ img_CentraSite_loginscreen.png </LoginProductLogo></pre>

Layout Component	Configuration Property	Description
Access as Guest (link)	<code>loginAsGuest</code>	<p>Contains property settings that are used to modify the Access as Guest link.</p> <p>If the attribute <code>loginAsGuest</code> is set to <code>true</code>, then the Login page is skipped and the Welcome page appears to users when accessing CentraSite Business UI. This allows anonymous users to skip the Login screen and access the registry anonymously. You can perform a login by clicking the Log in link in the masthead of CentraSite Business UI.</p> <p>If the attribute <code>loginAsGuest</code> is set to <code>false</code>, then the Login page appears to users when accessing the CentraSite Business UI.</p>
	<code>Guest visibility</code>	<p>Contains property settings that are used to modify the Access as Guest link.</p> <p>If the attribute <code>visibility</code> is set to <code>true</code>, then the Access as Guest link will appear in the Login page of CentraSite Business UI.</p> <p>If the attribute <code>visibility</code> is set to <code>false</code>, then the Access as Guest link will not appear in the user interface. This prevents anonymous users from skipping the Login page and accessing the registry anonymously.</p> <p>To enable or disable the Access as Guest link, configure the property statement:</p> <pre><Guest visibility="true" /></pre>
Remember me (check box)	<code>RememberMe</code>	<p>Contains property settings that are used to modify the Remember me check box.</p> <p>If the attribute <code>visibility</code> is set to <code>true</code>, then the Remember Me check box will appear in the Login page of CentraSite Business UI. This helps users to store the specified login credentials as a cookie and automatically use it the next time they access CentraSite Business UI.</p> <p>If the attribute <code>visibility</code> is set to <code>false</code>, then the Remember Me check box will not appear in the user interface. This enforces users to manually specify their login credentials every time they access CentraSite Business UI.</p>

Layout Component	Configuration Property	Description
		<p>To enable or disable the Remember me check box, configure the property statement:</p> <pre><RememberMe visibility="true" /></pre>
Request an Account (link)	RequestAnAccount	<p>Contains property settings that are used to modify the Request an Account link.</p> <p>If the attribute <code>visibility</code> is set to <code>true</code>, then the Request an Account link will appear in the Login page of CentraSite Business UI. This enables anonymous users to create or register a new user account in CentraSite registry.</p> <p>If the attribute <code>visibility</code> is set to <code>false</code>, then the Request an Account link will not appear in the user interface.</p> <p>To modify the Request an Account link, configure the property statement:</p> <pre><RequestAnAccount visibility="true"></pre>
	ReasonForRequest	<p>Contains property settings that are used to modify the Request an Account link.</p> <p>If the attribute <code>visibility</code> is set to <code>true</code>, then the Reason for Request text box will appear in the Create an Account page of CentraSite Business UI. This allows users to enter the specific reason why they require this account in CentraSite.</p> <p>If the attribute <code>visibility</code> is set to <code>true</code>, then the Reason for Request text box will not appear in the user interface.</p> <p>To enable or disable the Reason for Request text box, configure the property statement:</p> <pre><ReasonForRequest visibility="true"/></pre>
Take a Tour (button)	TakeATour	<p>Contains property settings that are used to modify the Take a Tour button.</p> <p>If the attribute <code>visibility</code> is set to <code>true</code>, then the Take a Tour button will appear in the Login page of CentraSite Business UI. This allows users to enter the specific reason why they require this account in CentraSite.</p>

Layout Component	Configuration Property	Description
		<p>If the attribute <code>visibility</code> is set to <code>false</code>, then the Take a Tour button will not appear in the user interface.</p> <p>To modify the Take a Tour button, configure the property statement:</p> <pre><TakeATour visibility="true" url="http://www.softwareag.com"> CS_MSG_INMBX_BTN_TOUR</TakeATour></pre>
Take a Tour (image)	TakeATour	<p>Contains property settings that are used to modify the Take a Tour image.</p> <p>If the attribute <code>visibility</code> is set to <code>true</code>, then the Take a Tour image will appear in the Login page of CentraSite Business UI. This allows users to enter the specific reason why they require this account in CentraSite.</p> <p>If the attribute <code>visibility</code> is set to <code>false</code>, then the Take a Tour image will not appear in the user interface.</p> <p>To modify the Take a Tour image, configure the property statement:</p> <pre><TakeATourImage> images/custom/ico_up_TakeATour_50x43.png </TakeATourImage></pre>

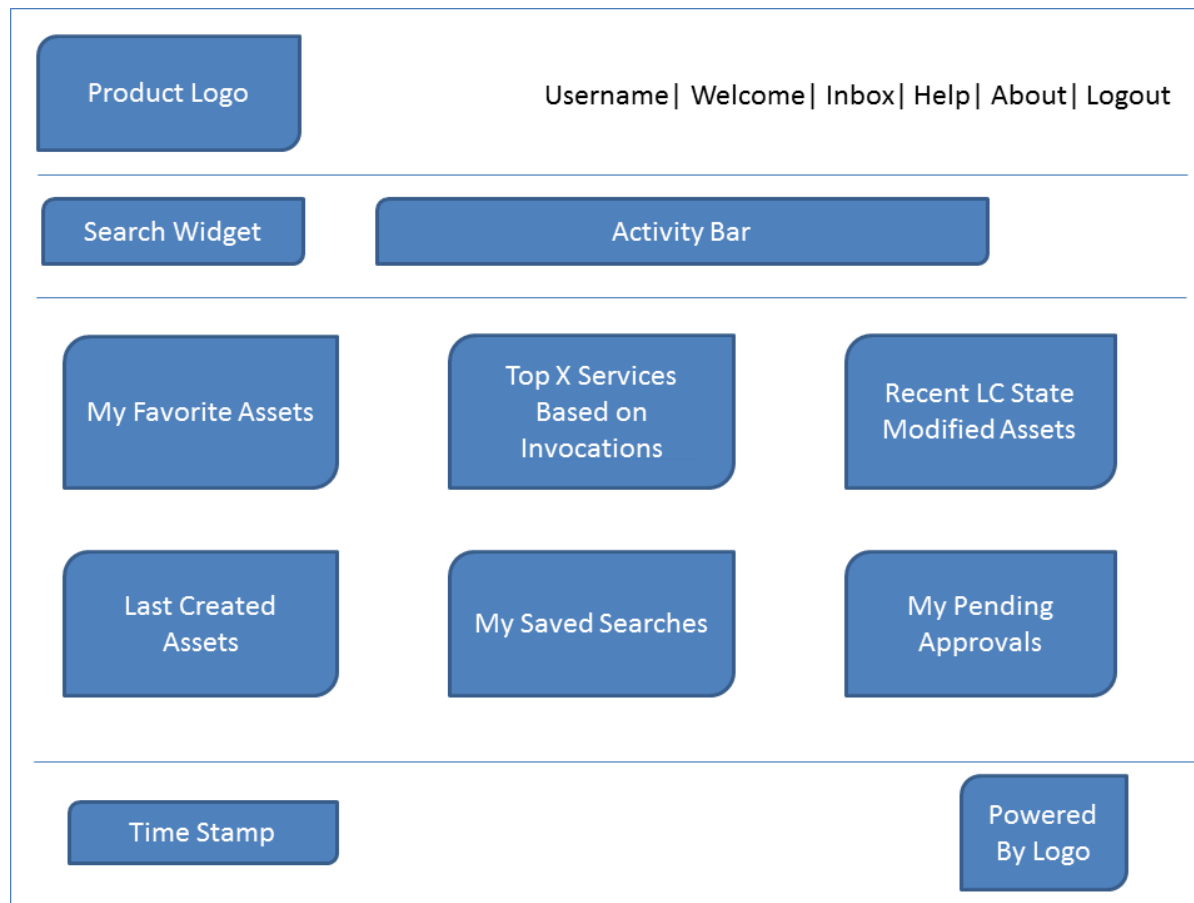
Footer Configuration

Layout Component	Configuration Property	Description
Copyright	CopyRight	<p>Contains property settings that are used to modify the Copyright information.</p> <p>If the attribute <code>visibility</code> is set to <code>true</code>, then the Copyright information will appear in the Login page of CentraSite Business UI.</p> <p>If the attribute <code>visibility</code> is set to <code>false</code>, then the copyright information will not appear in the user interface.</p> <p>To enable or disable the Copyright text, configure the property statement:</p>

Layout Component	Configuration Property	Description
		<pre><CopyRight visibility="true"> CS_MSG_INMBX_LBL_COPYRIGHT </CopyRight></pre>
Powered by Software AG	PoweredBy	<p>Contains property settings that are used to modify the Software AG logo.</p> <p>If the attribute <code>visibility</code> is set to <code>true</code>, then the Software AG logo will appear in the Login page of CentraSite Business UI.</p> <p>If the attribute <code>visibility</code> is set to <code>false</code>, then the Software AG logo will not appear in the user interface.</p> <p>To enable or disable the Powered by Software AG logo, configure the property statement:</p> <pre><PoweredBy tooltip="CS_MSG_INMBX_LBL_FOOTER_TEXT" visibility="true">images/system/ webfooter_powered_by.png </PoweredBy></pre>

Customizing CentraSite Welcome Page

The Welcome page that is visible when you access CentraSite Business UI can be customized to suit your requirements. You can change the search widget, search scopes, activity menus, the main navigation links, and the Welcome page. The standard Welcome page gives you quick links to the pages of CentraSite Business UI that you use frequently during your day-to-day work with CentraSite. It also provides links to external web sites that provide useful information related to CentraSite. A search box allows you perform a keyword search for registry assets whose name or description contains the given keyword. The figure depicts the schematic layout of the welcome page.



The header section displays the company logo and the product logo. You can change the logos and colors used to display them or hide the logos as part of the page customization. You can change the background color for the whole header section.

The navigation bar contains a search widget that allows you to perform a keyword search for an asset. You can have a user-defined search widget or the default search widget as part of the page customization. You can change the background color of the search widget. Additionally, a browse link is displayed by default that allows you to perform an advanced search for an asset. You can customize the browse link to contain a custom URL that either opens an Advanced Search page or an external page of your choice. You can rename or hide the browse link as part of the page customization.

The Activities menu includes options that enable you to access the various functions, such as Create Assets, Import Assets, and so on. You can rearrange the order of these menu options, customize the activities menu to include or exclude one or more functions as part of the customization process.

The Welcome page can contain one or more portlets. Each portlet contains a header and content. The header includes a title, with an icon adjacent to the text, some selectable markers (for example, set user configuration of an individual portlet, expand and collapse a portlet) and a close button. Under the header, you can have a list of entries, either representing the result set of a search query, any external HTML page or a graphical image. The rendering of the portlets in the welcome page depends on how you configure them.

CentraSite Business UI supports the following portlets:

- **Text portlet** – Represents the result set of a saved search query as executable actions. An action contains the URL of a search result (for example, the name of an asset) that renders the appropriate details page (in this case, the asset details page). The names of the search results are displayed as a table consisting of single or multiple columns. Each table cell contains one executable action and may have an icon beside it.
- **IFrame portlet** – Represents an arbitrary URL that points to the corresponding external HTML page inside the HTML IFrame component.
- **Graphical portlet** – Represents the saved search queries in graphical representation.

The footer section displays the current date and time, copyright information, and subtitle text. You can change the background color of the footer section, and hide the copyright information and the subtitle text as part of the page customization.

➤ To customize and install the customized welcome page

1. Stop Software AG Runtime
2. Customize the Welcome page configuration.

The standard configuration file **centrasite.xml** delivered with the CentraSite kit contains all of the names of the CentraSite Business UI extension points, including the extension point for the Welcome page configuration. You must create a copy of the extension point for the Welcome page configurations in the customization file, **centrasite.xml**, in order to define a customized LogWelcomein page. For more information, see [“CentraSite Welcome Page Configuration” on page 67](#).

You can find the customization file, **centrasite.xml**, on `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\custom\conf`.

3. Start Software AG Runtime.

The changes incorporated are now visible in the Welcome page.

CentraSite Welcome Page Configuration

The element `<GUIConfiguration>` in the **centrasite.xml** file contains the properties for customization of CentraSite Business UI. The Welcome page configuration includes configuring the header, navigation panel, content panel, and the footer.

Note:

Make sure that you specify an image file using its relative path in the `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\custom\images` directory. In a Windows environment, use forward slashes instead of backward slashes in the path name.

Header Configuration

Layout Component	Configuration Property	Description
CENTRASITE	ProductLogo	<p>Contains property settings that are used to modify the company logo in the Welcome page of CentraSite Business UI.</p> <p>To modify the company logo, configure the property statement:</p> <pre><ProductLogo tooltip= "CS_MSG_INMBX_LBL_PRODUCT_INTRO_HEADER"> images/custom/cs_product_logo_70x70.png </ProductLogo></pre>
Welcome	dashboard	<p>Contains property settings that are used to modify the Welcome title in the Welcome page of CentraSite Business UI.</p> <p>To modify the Welcome title, configure the property statement:</p> <pre><HeaderLink id="dashboard" displayName="CS_MSG_INMBU_LINK_DASHBOARD" token="welcome:welcome" /></pre>
Inbox	inbox	<p>Contains property settings that are used to modify the Inbox title in the Welcome page of CentraSite Business UI.</p> <p>To modify the Inbox title, configure the property statement:</p> <pre><HeaderLink id="inbox" displayName="CS_MSG_INMBU_LINK_INBOX" token="inbox:inbox" /></pre>
Help	help	<p>Contains property settings that are used to modify the Help title in the Welcome page of CentraSite Business UI.</p> <p>To modify the Help title, configure the property statement:</p> <pre><HeaderLink id="help" displayName="CS_MSG_INMBU_LINK_HELP" token="help:help" /></pre>

Layout Component	Configuration Property	Description
About	about	<p>Contains property settings that are used to modify the About title in the Welcome page of CentraSite Business UI.</p> <p>To modify the About title, configure the property statement:</p> <pre><HeaderLink id="about" displayName="CS_MSG_INMBU_LINK_ABOUT" token="about:about" /></pre>
	HeaderMenuSeparator	<p>To modify the separator, configure the property statement:</p> <pre><HeaderMenuSeparator tooltip= "CS_MSG_INMBX_LBL_HEADER_MENU_SEPERATOR"> images/custom/Separator_White_1X1.png </HeaderMenuSeparator></pre>

Important:

You cannot customize the header link **Log Out** (even if you belong to the CentraSite Administrator role).

Navigation Panel Configuration

Layout Component	Configuration Property	Description
TypeAhead Search	TypeAheadSearch	<p>Contains property settings that are used to modify the TypeAhead Search textbox in the Welcome page of CentraSite Business UI.</p> <p>To modify the TypeAhead Search textbox, configure the property statement:</p> <pre><TypeAheadSearch minOffsetToStartSearch="3" numOfResultsToShow="5" maxCharactersToShow="60" /></pre>
Browse (link)	BrowseLink	<p>Contains property settings that are used to modify the Browse link.</p> <p>If the attribute <code>visibility</code> is set to <code>true</code>, then the Browse link will appear in the Welcome page of CentraSite Business UI. This helps users to browse for assets in the CentraSite registry using the advanced search options.</p>

Layout Component	Configuration Property	Description
		<p>If the attribute <code>visibility</code> is set to <code>false</code>, then the Browse link will not appear in the user interface. This enforces users to manually specify the keywords in the TypeAhead Search textbox.</p> <p>You can customize the Browse link in the following ways:</p> <ul style="list-style-type: none">■ Enable or disable the Browse link■ Provide a custom URL■ Rename the Browse link <p>To modify the details of the Browse link, configure the property statement:</p> <pre><BrowseLink visibility="true" displayName="CS_MSG_INMBU_LINK_BROWSE" token="browse:browse" /></pre>
<ul style="list-style-type: none">■ Everything■ Assets	<code>SearchScopes</code>	<p>Contains property settings that are used to modify the list of search scopes in the Welcome page of CentraSite Business UI.</p> <p>To modify the list of search scopes, configure the property statement:</p> <pre><SearchScopes> <SearchScope id="Everything" isExpandable="true" class= "com.softwareag.centrasite.api.csom. search.impl.EverythingScope" exclude= "uddi:7613515f-77eb-11dd-bc9f-f62b6cf80b00"> INMCL_STR_Everything</SearchScope> <SearchScope id="Assets" isExpandable="false" class= "com.softwareag.centrasite.api.csom. search.impl.AssetScope" exclude= "uddi:7613515f-77eb-11dd-bc9f-f62b6cf80b00"> INMCL_STR_Assets</SearchScope> </SearchScopes></pre>
Search (icon)	<code>SearchIcon</code>	<p>Contains property settings that are used to modify the Search icon in the Welcome page of CentraSite Business UI.</p> <p>To modify the Search icon, configure the property statement:</p> <pre><SearchIcon>images/custom/SearchButton.png</SearchIcon></pre>

Layout Component	Configuration Property	Description
■ Create Asset	Activities	Contains property settings that are used to modify the list of activities in the Welcome page of CentraSite Business UI.
■ Generate Reports		To modify the list of activities, configure the property statement:
■ Manage Organizations		<pre> <Activities> <Activity id="Create Assets" class="com.softwareag.centrasite.api. activity.impl. CreateAssetActivityImpl"> INMCL_ACTIVITY_CREATE_ASSET </Activity> <Activity id="Global Reports" class="com.softwareag.centrasite.api. activity.impl. GlobalReportActivityImpl"> INMCL_ACTIVITY_GLOBAL_REPORTS </Activity> <Activity id="Manage Organizations" class="com.softwareag.centrasite.api. activity.impl. ManageOrganizationsActivityImpl"> INMCL_ACTIVITY_MANAGE_ORGANIZATIONS </Activity> <Activity id="ManageGovernanceRules" class="com.softwareag.centrasite.api. activity.impl. ManageGovernanceRulesActivityImpl"> INMCL_ACTIVITY_MANAGE_GOVERNANCE_RULES </Activity> <Activity id="ManageTaxonomies" class="com.softwareag.centrasite.api. activity.impl. ManageTaxonomiesActivityImpl"> INMCL_ACTIVITY_MANAGE_TAXONOMIES </Activity> <Activity id="Asset Navigator" class="com.softwareag.centrasite.api. activity.impl. AssetNavigatorActivityImpl"> INMCL_ACTION_ASSETNAVIGATOR </Activity> </Activities> </pre>
■ Manage Governance Rules		
■ Manage Taxonomies		
■ Asset Navigator		

Content Panel Configuration

Layout Component	Configuration Property	Description
Portlets	Portlets	Contains property settings that are used to modify the list of portlets in the Welcome page of CentraSite Business UI.

Layout Component	Configuration Property	Description
		<p>To modify the list of portlets, configure the property statement:</p> <pre><Portlets> <Portlet id="MyFavoritesPortlet" description="INMBU_STR_PORTLET_FAVORITES_DESC" type="text" dataFeed="MyFavorites" icon="images/system/favorites_16X16.png" row="0" column="0" isVisible="true" refreshInterval="0" actions= "configure,refresh"> INMBU_STR_PORTLET_NAME_MY_FAVORITES </Portlet> ... </Portlets></pre>
PortletActions	Portlets	<p>Contains property settings that are used to modify the list of actions in each portlet in the Welcome page of CentraSite Business UI.</p> <p>To modify the list of portlet actions, configure the property statement:</p> <pre><PortletActions> <PortletAction id="configure"> INMBU_STR_PORTLET_ACTION_EDIT </PortletAction> <PortletAction id="refresh"> INMBU_STR_PORTLET_ACTION_REFRESH </PortletAction> </PortletActions></pre>

Footer Configuration

Layout Component	Configuration Property	Description
Time	Time	<p>Contains property settings that are used to modify Time.</p> <p>If the attribute <code>visibility</code> is set to <code>true</code>, then the Time will appear in the Welcome page of CentraSite Business UI.</p> <p>If the attribute <code>visibility</code> is set to <code>false</code>, then the Time will not appear in the user interface.</p> <p>To enable or disable Time, configure the property statement:</p> <pre><Time visibility="true" /></pre>
Powered by Software AG	PoweredBy	<p>Contains property settings that are used to modify the Software AG logo.</p>

Layout Component	Configuration Property	Description
		<p>If the attribute <code>visibility</code> is set to <code>true</code>, then the Software AG logo will appear in the Welcome page of CentraSite Business UI.</p> <p>If the attribute <code>visibility</code> is set to <code>false</code>, then the Software AG logo will not appear in the user interface.</p> <p>To enable or disable the Powered by Software AG logo, configure the property statement:</p> <pre><PoweredBy tooltip="CS_MSG_INMBX_LBL_FOOTER_TEXT" visibility="true">images/system/ webfooter_powered_by.png </PoweredBy></pre>

Customizing CentraSite Content Page

To customize content pages, you can use extension points for the CentraSite Business UI and implement Java classes and methods to plug in to the extension points.

An extension point is characterized by:

- An ID with which it can be referenced.
- A GWT widget that wraps the extension functionality.

You can find the sample code for defining the extension points in the file `ExtensionViewFactory.java`. You can find the file in the demos folder under the `<CentraSiteInstallDirectory>`.

Log on to CentraSite Business UI

Usage	Use at the start of a session of CentraSite Business UI.
Elements	<ul style="list-style-type: none"> ■ <code>element name=ExtensionPointLogin</code> ■ Value of attribute <code>custom</code> (true or false)
Processing	When you log on to CentraSite Business UI, a user-defined Login page is rendered.
Provided by	<i>BusinessUI</i>
Example	<code><ExtensionPointLogin custom="false" /></code>

Request an Account

Usage	Use this when requesting an account in CentraSite Business UI.
--------------	--

Elements	■ <code>element name=ExtensionPointRegister</code>
	■ Value of attribute <code>custom</code> (true or false)
Processing	When you request an account in CentraSite, this invokes a user-defined registration page that displays parameters required for entering the user credentials.
Provided by	<i>BusinessUI</i>
Example	<code><ExtensionPointRegister custom="false" /></code>

Extending Activity Menu

The Activity menu contains a set of functions that you can perform through CentraSite Business UI. The Activity menu helps you access the following functions by default:

- Create Asset
- Generate Reports
- Manage Organizations
- Manage Governance Rules
- Manage Taxonomies
- Asset Navigator

You can add a custom activity to have the specific function and format that you require. Additionally, you can change the order of displaying activities within the activity menu.

➤ To extend the activity menu for additional functions

1. Implement the custom activity as a GWT extension point. For procedures, see [“Implement Activity as an Extension” on page 74](#).
2. Configure the custom activity in **centrasite.xml** configuration file. For procedures, see [“Enable Activity through Configuration” on page 75](#).

Implement Activity as an Extension

Usage	Use this to define a custom activity. For example, add an activity, reorder the existing activities, and remove activities to suit your requirements.
Elements	■ <code>id</code>
	■ <code>class</code>
Interfaces	■ <code>ActivityManager</code>

■ ActivityMenu

Abstract base class	Activity
Processing	When you start the user interface, a custom defined activity menu is rendered.
Provided by	<i>CentraSiteLogicLayer</i>
Code	See the sample code.

You can find sample code for defining the custom activity as an extension point in the file `CentraSiteBUIExtension.gwt.xml`. You can find the file in the `demos` folder in the `<CentraSiteInstall_Directory>`.

```
<!-- Use ExtensionWidgetFactory by default -->
<replace-with
  class="com.softwareag.centrasite.bui.extension.client.
    factory.ExtensionWidgetFactory">
<when-type-is
  class="com.softwareag.centrasite.bui.extension.core.
    client.IExtensionWidgetFactory"/>
</replace-with>
```

Enable Activity through Configuration

After you define a custom activity as an extension you have to enable the activity configuration so as to display the activity in CentraSite Business UI.

Important:

The activity configuration parameters initially defined in the configuration file are editable and cannot be protected.

➤ To enable the custom activity configuration

1. Open the customization file, `centrasite.xml`, in a rich text editor.

You can find the **centrasite.xml** file on `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\custom\conf`.

2. Locate Activities Configuration under `<CLL Configurations>`.

The Activities snippet looks like this:

```
<Activities>
  <Activity id="Create Assets"
    class="com.softwareag.centrasite.api.
      activity.impl.CreateAssetActivityImpl">
    INMCL_ACTIVITY_CREATE_ASSET
  </Activity>
  <Activity id="Global Reports"
    class="com.softwareag.centrasite.api.
      activity.impl.GlobalReportActivityImpl">
```

```
    INMCL_ACTIVITY_GLOBAL_REPORTS
  </Activity>
  <Activity id="Manage Organizations"
    class="com.softwareag.centrasite.api.
    activity.impl.ManageOrganizationsActivityImpl">
    INMCL_ACTIVITY_MANAGE_ORGANIZATIONS
  </Activity>
  <Activity id="ManageGovernanceRules"
    class="com.softwareag.centrasite.api.
    activity.impl.ManageGovernanceRulesActivityImpl">
    INMCL_ACTIVITY_MANAGE_GOVERNANCE_RULES
  </Activity>
  <Activity id="ManageTaxonomies"
    class="com.softwareag.centrasite.api.
    activity.impl.ManageTaxonomiesActivityImpl">
    INMCL_ACTIVITY_MANAGE_TAXONOMIES
  </Activity>
  <Activity id="Asset Navigator"
    class="com.softwareag.centrasite.api.
    activity.impl.AssetNavigatorActivityImpl">
    INMCL_ACTION_ASSETNAVIGATOR
  </Activity>
</Activities>
```

3. Append the property configuration statement for custom activity. For example, Create Report Templates.

```
<Activities>
  <Activity id="Create Report Templates"
    class="com.softwareag.centrasite.api.
    activity.impl.CreateReportTemplatesImpl">
    INMCL_ACTIVITY_CREATE_REPORT _TEMPLATES
  </Activity>
</Activities>
```

The input parameters are:

Parameter	Denotes
id	A unique identifier for the activity, in this example for Create Report Templates. It uniquely distinguishes an activity in the CentraSite registry. If you wish to reset the activity at a later stage, you identify the activity using this id.
class	The implementation class of <code>com.softwareag.centrasite.api.activity.Activity</code> interface.
INMCL_ACTIVITY	The i18n string to display the activity in CentraSite Business UI. If you want to specify a localized name, type the message ID starting with a prefix INMCL (in this example, <code>INMCL_ACTIVITY_CREATE_REPORT_TEMPLATES</code>). The activity internally identifies the message ID and displays the localized name, if available in the message database. Else, if you specify a name without INMBU prefix, the activity will simply display a plain text name.

The Activities snippet now looks similar to this.

```
<Activities>
  <Activity id="Create Assets"
    class="com.softwareag.centrasite.api.
    activity.impl.CreateAssetActivityImpl">
    INMCL_ACTIVITY_CREATE_ASSET
  </Activity>
  <Activity id="Global Reports"
    class="com.softwareag.centrasite.api.
    activity.impl.GlobalReportActivityImpl">
    INMCL_ACTIVITY_GLOBAL_REPORTS
  </Activity>
  <Activity id="Manage Organizations"
    class="com.softwareag.centrasite.api.
    activity.impl.ManageOrganizationsActivityImpl">
    INMCL_ACTIVITY_MANAGE_ORGANIZATIONS
  </Activity>
  <Activity id="ManageGovernanceRules"
    class="com.softwareag.centrasite.api.
    activity.impl.ManageGovernanceRulesActivityImpl">
    INMCL_ACTIVITY_MANAGE_GOVERNANCE_RULES
  </Activity>
  <Activity id="ManageTaxonomies"
    class="com.softwareag.centrasite.api.
    activity.impl.ManageTaxonomiesActivityImpl">
    INMCL_ACTIVITY_MANAGE_TAXONOMIES
  </Activity>
  <Activity id="Asset Navigator"
    class="com.softwareag.centrasite.api.
    activity.impl.AssetNavigatorActivityImpl">
    INMCL_ACTION_ASSETNAVIGATOR
  </Activity>
  <Activity id="Create Report Templates"
    class="com.softwareag.centrasite.api.
    activity.impl.CreateReportTemplatesImpl">
    INMCL_ACTIVITY_CREATE_REPORT_TEMPLATES
  </Activity>
</Activities>
```

4. Locate Activities Configuration under <UIProperties>.

The configuration snippet looks like this:

```
<Activities noOfActivitiesInMainNav="5">
  <Activity id="CreateAssetBui" refId="Create Assets" numOfProfilesPerRow="3"
    token="activity:Create_Assets" />
  <Activity id="GlobalReportsBui" refId="Global Reports"
    token="activity:Global_Reports" />
  <Activity id="ManageOrganizationBui" refId="Manage Organizations"
    token="activity:Manage_Organizations" />
  <Activity id="ManageGovernanceRulesBui" refId="ManageGovernanceRules"
    token="activity:governance_rules" />
  <Activity id="ManageTaxonomiesBui" refId="ManageTaxonomies"
    token="activity:manage_taxonomies" />
  <Activity id="Asset Navigator" refId="Asset Navigator"
    token="activity:Asset_Navigator" />
</Activities>
```

- Append the defined custom activity (for example, Create Report Templates) configuration statement.

```
<Activity id="Create Report Templates"
  token="customactivity:Create Report Templates"/>
```

The input parameters are:

Parameter	Denotes
refId	Refers to an unique identifier id for the activity, as defined in the CLL Configurations.
token	The place tokenizer of the activity that helps in rendering the activity as a deep link URL. Important: For rendering any custom activity, the place tokenizer should contain the customactivity: prefix. Thus for example, a custom activity Create Report Templates would have the place tokenizer read as customactivity:Create Report Templates.

The Activities now looks similar to this.

```
<Activities noOfActivitiesInMainNav="5">
  <Activity id="CreateAssetBui" refId="Create Assets" numOfProfilesPerRow="3"
    token="activity:Create_Assets" />
  <Activity id="GlobalReportsBui" refId="Global Reports"
    token="activity:Global_Reports" />
  <Activity id="ManageOrganizationBui" refId="Manage Organizations"
    token="activity:Manage_Organizations" />
  <Activity id="ManageGovernanceRulesBui" refId="ManageGovernanceRules"
    token="activity:governance_rules" />
  <Activity id="ManageTaxonomiesBui" refId="ManageTaxonomies"
    token="activity:manage_taxonomies" />
  <Activity id="Asset Navigator" refId="Asset Navigator"
    token="activity:Asset_Navigator" />
  <Activity id="Create Report Templates" refId="Create Report Templates"
    token="customactivity:Create_Report_Templates"/>
</Activities>
```

- Save and close the configuration file.
- Restart Software AG Runtime.

Replace Standard Search Widget

Usage	Use this when the layout of a standard search widget (comprising of search field, search options, and search execution) needs to be replaced with custom search widget.
--------------	---

Elements	■ <code>element name=ExtensionPointSearchWidget</code>
	■ Value of attribute <code>custom</code> (true or false)
Processing	When you start the user interface, a user-defined (keyword) search widget is rendered.
Provided by	<i>BusinessUI</i>
Example	<code><ExtensionPointSearchWidget custom="false" /></code>

Advanced Search Criteria

Usage	Extend the existing advanced search criteria of an asset keyword search performed by typing custom related search settings, for example, you can add specific search predicates for your own object types.
Elements	■ <code>element name=ExtensionPointSearchCriteria</code>
	■ Value of attribute <code>custom</code> (true or false)
Processing	■ This extension is initialized through the <code>Extension.showSearchResults()</code> method.
	■ Returns a user-defined advanced search (layout) screen displaying the available search options.
	■ Refreshes the default search results page.
Provided by	<i>BusinessUI</i>
Example	<code><ExtensionPointSearchCriteria custom="false" /></code>

Browse Link

Usage	■ Change the text string Browse displayed in the default view.
	■ Change the URL of the Browse link to contain either a page of your choice within the CentraSite Business UI, or to an external web page that you regularly visit within the context of your work with CentraSite.
Elements	■ <code>element name=ExtensionPointBrowseView</code>
	■ Value of attribute <code>custom</code> (true or false)
Processing	Browse link that would either open a user-defined Search Results page or a custom link that would open a CentraSite page or an external web page of your choice.
Provided by	<i>BusinessUI</i>

Example	<code><ExtensionPointBrowseView custom="false" /></code>
----------------	--

Browse Search Criteria

Usage	Extend the existing advanced search panel of an asset browse for typing custom related search settings, for example, you can add specific search predicates for your own object types.
--------------	--

Elements	<ul style="list-style-type: none">■ <code>element name=ExtensionPointBrowseCriteria</code>■ Value of attribute <code>custom</code> (true or false)
-----------------	---

Processing	<ul style="list-style-type: none">■ This extension is initialized through the <code>Extension.showBrowseResults()</code> method.■ Returns a user-defined advanced search (layout) screen showing the available search options.■ Refreshes the default search results page.
-------------------	--

Provided by	<i>BusinessUI</i>
--------------------	-------------------

Example	<code><ExtensionPointBrowseCriteria custom="false" /></code>
----------------	--

Extend Search View

Usage	Change the default layout of entire search view to suit your requirements.
--------------	--

Elements	<ul style="list-style-type: none">■ <code>element name=ExtensionPointSearch</code>■ Value of attribute <code>custom</code> (true or false)
-----------------	---

Processing	<p>When you want to search assets, this invokes:</p> <ul style="list-style-type: none">■ a user-defined advanced search (layout) screen showing the available search options.■ a user-defined search results (layout) screen displaying all assets that fit the search criteria.
-------------------	---

Provided by	<i>BusinessUI</i>
--------------------	-------------------

Example	<code><ExtensionPointSearch custom="false" /></code>
----------------	--

Replace Standard Asset Detail Page

Usage	Use this to customize the layout of a standard asset details page to fit your needs.
--------------	--

Elements	<ul style="list-style-type: none">■ <code>element name=ExtensionPointAssetDetails</code>
-----------------	--

- Value of attribute custom (true or false)

Processing	If the details page for an asset is opened, an user-defined asset details page is rendered.
Provided by	<i>BusinessUI</i>
Example	<code><ExtensionPointAssetDetails custom="false" /></code>

Extending Action Menu

The action bar in the CentraSite Business UI allows a user to execute a defined set of actions on the selected objects. You can perform various actions such as modifying object details, attaching documents, generating reports, and so on, for a single object (through the object details page) or a set of objects (Search Results page).

CentraSite Business UI has a predefined set of actions available for the registry objects, for example, assets, policies, and gateways. In addition to the predefined actions, you can also define your own custom actions for the objects.

Additionally, you can customize the way in which you want the action to render:

- You can add or remove an action to or from the action menu.
- You can change the text string displayed for an action in the action menu.
- You can change the icon that represents an action in the action menu.
- You can enable an action for bulk operation.

➤ To add an action menu

1. Implement the custom action as a GWT extension point. For procedures, see [“Implement Action as an Extension” on page 81](#).
2. Configure the custom action in **centrasite.xml** configuration file. For procedures, see [“Enable Action through Configuration” on page 82](#).

Implement Action as an Extension

Usage	Use this to define a custom action in CentraSite Business UI. For example, add an action, reorder the existing actions, enable or disable an action, and remove actions to suit your requirements.
Elements	<ul style="list-style-type: none"> ■ String <code>actionId</code> ■ String <code>commaSeparatedAssetId</code> ■ String <code>positionParam</code>

Interface	See the sample code
Abstract base class	ActionListener
Processing	When a list of actions for an action menu is retrieved, the following steps are performed for each known action: <ul style="list-style-type: none">■ create an instance of class.■ implement the method <code>processAction</code> of type <code>String</code>.
Provided by	<i>BusinessUI</i>
Code	See the sample code.

You can find sample code for defining the custom action as an extension point in the file `CentraSiteBUIExtension.gwt.xml`. You can find the file in the `demOs` folder in the `<CentraSiteInstall_Directory>`.

```
<!-- Use ActionListener by default -->
<replace-with
  class="com.softwareag.centrasite.bui.extension.client.action.ActionListener">
<when-type-is
  class="com.softwareag.centrasite.bui.extension.core.client.IActionListener"/>
</replace-with>
```

Enable Action through Configuration

After you define a custom action as an extension, you have to enable the custom action configuration so as to display the action in the CentraSite Business UI.

Important:

The action configuration parameters initially defined in the configuration file are editable and cannot be protected.

➤ To enable an action through configuration

1. Open the customization file, `centrasite.xml`, in a rich text editor.

You can find the **centrasite.xml** file on `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\custom\conf`.

2. Locate Actions under `<CLL Configuration>`.

The Actions snippet looks like this:

```
<Actions>
<Action id="SaveAction" inbox="false" bulk="false"
  group="basic"
  icon="images/custom/actionbar/01_Save.gif"
  class="com.softwareag.centrasite.api.csom.action.impl.
  SaveActionImpl">INMCL_ACTION_SAVE
```

```

</Action>
<Action id="EditAction" inbox="false" bulk="false"
  group="basic"
  icon="images/custom/actionbar/02_Edit.gif"
  class="com.softwareag.centrasite.api.csom.action.impl.
  EditActionImpl">INMCL_ACTION_EDIT
</Action>
.....
.....
<Action id="RuntimeAliasAction" inbox="false" bulk="true"
  group="addRuntimeAction" applicable="policy"
  icon="images/custom/actionbar/managerRuntimeAlias.png"
  class="com.softwareag.centrasite.api.csom.action.runtime.impl.
  ManageRuntimeAliasActionImpl">INMCL_ACTION_AddRuntimeAlias
</Action>
<Action id="AddGateway" inbox="false" bulk="true"
  group="addRuntimeAction" applicable="policy"
  icon="images/custom/actionbar/add_gateway_32X32.png"
  class="com.softwareag.centrasite.api.csom.action.runtime.impl.
  ManageGatewayActionImpl">INMCL_ACTION_AddGateway
</Action>
</Actions>

```

3. Append the property configuration statement for custom action. For example, Change Owner.

```

<Action id="ChangeOwnerAction" inbox="false" bulk="false"
  group="basic"
  icon="images/custom/actionbar/ChangeOwner.gif"
  class="com.softwareag.centrasite.api.csom.action.impl.
  ChangeOwnerActionImpl">INMCL_ACTION_CHANGE_OWNER
</Action>

```

The input parameters are:

Parameter	Denotes
Action id	A unique identifier for the action, in this example ChangeOwnerAction. It uniquely distinguishes an action in the CentraSite registry. If you want to reset the action at a later stage, you identify the action using this id.
class	The implementation class of interface: com.softwareag.centrasite.api.csom.action.impl.ChangeOwnerActionImpl
INMCL_ACTIVITY	The i18n string to display the action in CentraSite Business UI. If you want to specify a localized name, type the message ID starting with a prefix INMBU (in this example, INMCL_ACTION_CHANGE_OWNER). The action internally identifies the message ID, and display the localized name, if available in the message database. Else, if you specify a name without INMBU prefix, the action displays a plain text name.

The Actions snippet now looks similar to this.

```
<Actions>
```

```
<Action id="SaveAction" inbox="false" bulk="false"
  group="basic"
  icon="images/custom/actionbar/01_Save.gif"
  class="com.softwareag.centrasite.api.csom.action.impl.
  SaveActionImpl">INMCL_ACTION_SAVE
</Action>
<Action id="EditAction" inbox="false" bulk="false"
  group="basic"
  icon="images/custom/actionbar/02_Edit.gif"
  class="com.softwareag.centrasite.api.csom.action.impl.
  EditActionImpl">INMCL_ACTION_EDIT
</Action>
.....
<Action id="ChangeOwnerAction" inbox="false" bulk="false"
  group="basic"
  icon="images/custom/actionbar/ChangeOwner.gif"
  class="com.softwareag.centrasite.api.csom.action.impl.
  ChangeOwnerActionImpl">INMCL_ACTION_CHANGE_OWNER
</Action>
.....
<Action id="RuntimeAliasAction" inbox="false" bulk="true"
  group="addRuntimeAction" applicable="policy"
  icon="images/custom/actionbar/managerRuntimeAlias.png"
  class="com.softwareag.centrasite.api.csom.action.runtime.impl.
  ManageRuntimeAliasActionImpl">INMCL_ACTION_AddRuntimeAlias
</Action>
<Action id="AddGateway" inbox="false" bulk="true"
  group="addRuntimeAction" applicable="policy"
  icon="images/custom/actionbar/add_gateway_32X32.png"
  class="com.softwareag.centrasite.api.csom.action.runtime.impl.
  ManageGatewayActionImpl">INMCL_ACTION_AddGateway
</Action>
</Actions>
```

4. To enable an action entry for bulk usage, change `false` to `true`. Similarly, if the action is already available for bulk usage and you want to revert, set the value to `false`.
5. Save and close the configuration file.
6. Restart Software AG Runtime.

Sample Custom Actions

CentraSite includes the following sample custom actions:

- Change Owner Action
- Runtime Policy Action

The sample custom actions are available in the `demos` folder in the `<CentraSiteInstall_Directory>`.

Change Owner Action

The Change Owner Action does not have any UI associated with it. This action is applicable for both single and multiple objects. When the action is selected, the ownership of the selected objects

are automatically transferred to the current logged in user provided the logged in user has the CentraSite Administrator role.

Runtime Policy Action

The Runtime Policy Action has a UI associated with it. This action is applicable only for a single object. When the action is selected in the Asset Details page, CentraSite displays a list of policies that are applicable for the selected asset, and the user can select a policy for the execution. When the user selects the policy, the policy is executed, and the results of the execution are displayed to the user.

To create a custom action in CentraSite Business UI, the following classes or interfaces are required:

Interface CSOAction

- If you are writing a custom action, you must create a class implementing the CSOAction interface. The fully qualified name of this class must be mentioned in the element `<Action>` in the **centrasite.xml** file.
- The `ChangeOwnerAction.java` contains the implementation for the **Change Owner** action.

You can find the `ChangeOwnerAction.java` file in the folder `changeowner` in the `<CentraSiteInstall_Directory>\demos\CentraSiteBUIExtension\BusinessUIActions\WithoutUI\src\com\softwareag\centrasite\demos\bui\actions` directory.

- The `RunPolicyAction.java` is the implementation class for the **Execute Policy** action.

You can find the `RunPolicyAction.java` file in the folder `runpolicy` in the `<CentraSiteInstall_Directory>\demos\CentraSiteBUIExtension\BusinessUIActions\WithoutUI\src\com\softwareag\centrasite\demos\bui\actions` directory.

Class ActionInfo

- CentraSite Business UI makes use of GWT (Google Web Toolkit) to for the UI. To transmit data between the client and server side applications in GWT, the object must be serializable. The `ActionInfo` is a serializable class that holds the data from the class `CSOAction` such as the action ID, current action state, and so on. It also holds additional data, such as the selected assets on which the action has to be performed and any other inputs that are provided by the user. If your custom action does not require any user input other than the selected assets, then you can make use of the `ActionInfo` class, otherwise you have to create your own class inheriting the `ActionInfo` class.
- The **Change Owner** action does not need any other information. Therefore the `ActionInfo` implementation class itself is used for it.
- The **Execute Policy** action has additional information that has to be presented to the user and to be set in the `CSOAction` implementation class. Therefore, it uses the class `RunPolicyActionInfo.java` that is inheriting the `ActionInfo` class.

Interface IActionConverter

- This handles the conversion of the data from class `CSOAction` to class `ActionInfo` and vice versa. There must be a class implementing this interface. If your action does not require any new data other than the selected objects for executing the action, then we provide a default converter

class, called as `DefaultActionConverter`. On the other hand, you require additional inputs from the user, then they must be handled separately. In this case, you must create your own `IActionConverter` class and specify the same in the `ActionConverterFactory.getActionConverter()` method so that this converter is used for your action.

- The **Change Owner** action uses the class `DefaultActionConverter` as it does not have any new data to be converted.
- The **Execute Policy** action uses the class `RunPolicyActionConverter.java` to handle the conversion of the additional data.

Interface `IActionHandler`

- This handles the UI for the action. If your action does not have a UI, then we provide a default handler class, called as `DefaultActionHandler`. If your action requires a UI, then you have to implement the `IActionHandler` interface and handle the UI using this class based on the action states. The action handler for your action must be specified in the `ActionHandlerFactory.getActionHandler()` method.
- The **Change Owner** action does not have any UI associated with it, and uses the class `DefaultActionHandler`.
- The **Execute Policy** action has a UI, and uses the class `RunPolicyActionHandler.java`.

Once the above classes are ready and available to use, you would compile them, and then deploy the extensions. You will include your custom action in the **centrasite.xml** file as follows.

```
<Action id="{CustomActionName}" inbox="{true|false}" bulk="{true|false}"
group="{group_name}" icon='{URL/relative_path}'
class="fully.qualified.class.name">{Custom_Action_Name}</Action>
```

The input parameters are:

Parameter	Description
<code>id="{CustomActionName}"</code>	Name of the custom action. The name should be unique as it would be referenced throughout the action execution.
<code>inbox="{true false}"</code>	Enable or disable the visibility of the action in the Actions bar of the Inbox .
<code>bulk="{true false}"</code>	Enable or disable the visibility of the action in the Search Results page. If the value of this parameter is set to <code>false</code> , then the visibility of the action is restricted to the Asset Details page.
<code>group="{group_name}"</code>	A grouping for your action. This will help to group the actions according to the group name. By default, you can provide "basic" as the grouping.

Parameter	Description
<code>icon='{URL/relative_path}'</code>	<p>The URL of the icon that is to be used to represent this action in the Actions bar in CentraSite Business UI.</p> <p>You can specify the URL for the icon, or if the icon is located on your file system specify the file location relative to the <code><CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI</code> directory.</p>
<code>class="fully.qualified.class.name"</code>	The fully qualified class name of the implementation class <code>CSOAction</code> .
<code>{Custom_Action_Name}</code>	Display name of the action in the Actions bar in CentraSite Business UI.

Adding Portlets

Portlets enable you to create shortcuts and access the frequently-used functions, for example, Most Popular Assets, Recent Searches, and so on.

The Welcome page contains a default set of portlets, where each portlet represents a collection of predefined entries fetched using a saved search query.

You can define portlets to suit your needs and configure the tailored portlet to display in the Welcome page.

Additionally, you can customize the way in which you want portlet to render:

- You can add a portlet of type custom.
- You can define the configuration settings of the custom portlet.
- You can specify a display icon to represent the custom portlet in the Welcome page.
- You can make the custom portlet visible or invisible in the Welcome page.
- You can specify a location for displaying the custom portlet in the Welcome page.
- You can set the refresh interval for the custom portlet.

➤ To add portlets

1. Implement the custom portlet as a GWT extension point. For procedures, see [“Implement Portlet as an Extension” on page 88](#).
2. Configure the custom portlet in **centrasite.xml** configuration file. For procedures, see [“Enable Portlet through Configuration” on page 89](#).

Implement Portlet as an Extension

Usage	Use this to define a custom portlet in CentraSite Business UI.
Elements	<div>renderPortletContent</div> <ul style="list-style-type: none">■ String portletId■ String contentId■ String commaSeparatedParam <div>executePortletAction</div> <ul style="list-style-type: none">■ String portletId■ String actionId■ String contentId■ String commaSeparatedParam
Abstract base class	IPortletListener
Processing	<p>When the Welcome page containing one or more custom type portlets is loaded in the CentraSite Business UI, the following steps are performed for each custom portlet:</p> <ul style="list-style-type: none">■ create an instance of class■ implement the method renderPortletContent (within the class instance) to render each custom portlet defined in the configuration file.■ implement the method executePortletAction (within the class instance) to render each action of a custom portlet defined in the configuration file.
Provided by	<i>BusinessUI</i>
Code	See the sample code.

You can find sample code for defining the custom portlet as an extension point in the file `CentraSiteBUExtension.gwt.xml`. You can find the file in the `demos` folder in the `<CentraSiteInstall_Directory>`.

```
<!-- Use PortletListener by default -->
<replace-with
class="com.softwareag.centrasite.bui.extension.client.portlet.PortletListener">
  <when-type-is
class="com.softwareag.centrasite.bui.extension.core.client.IPortletListener"/>
</replace-with>
```

Important:

On a custom portlet defined as GWT extension point, whenever you execute a basic action (for example, Expand, Collapse, Close) through the user interface, CentraSite internally sends a corresponding `actionId` to the GWT extension.

Enable Portlet through Configuration

After you define a custom portlet as an extension, you have to enable the custom portlet configuration so as to display the portlet in the CentraSite Business UI.

Important:

The portlet configuration parameters initially defined in the configuration file are editable and cannot be protected.

➤ To enable a portlet through configuration

1. Open the customization file, `centrasite.xml`, in a rich text editor.

You can find the **centrasite.xml** file on `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\custom\conf`.

2. Locate the Portlet Configuration under BUI Configuration.

This is a sample configuration snippet.

```
<Portlets>
  <Portlet id="MyFavoritesPortlet"
    description="INMBU_STR_PORTLET_FAVORITES_DESC"
    type="text"
    dataFeed="MyFavorites"
    icon="images/system/favorites_16X16.png"
    row="0"
    column="0"
    isVisible="true"
    refreshInterval="0"
    actions="configure,refresh">
    INMBU_STR_PORTLET_NAME_MY_FAVORITES
  </Portlet>
  <Portlet id="LastCreatedAssetsPortlet"
    description="INMBU_STR_PORTLET_RECENTLY_CREATED_ASSETS_DESC"
    type="text"
    dataFeed="LastCreatedAssets"
    icon="images/system/recently_created_16X16.png"
    row="1"
    column="0"
    isVisible="true"
    refreshInterval="0"
    actions="configure,refresh">
    INMBU_STR_PORTLET_NAME_LAST_CREATED_ASSETS
  </Portlet>
  ...
</Portlets>
```

3. Append the required custom portlet (for example, My Custom Portlet) configuration statement.

```
<Portlet id="MyCustomPortlet"
description="INMBU_STR_PORTLET_CUSTOM_DESC"
type="text"
dataFeed="MyFavorites"
icon="images/system/custom_16X16.png"
row="1"
column="1"
isVisible="true"
refreshInterval="0"
actions="configure,refresh">
INMBU_STR_PORTLET_NAME_MY_CUSTOM_PORTLET
</Portlet>
```

wherein,

Parameter	Description
id [Portlet]	A unique identifier for the portlet. It uniquely distinguishes a portlet in the CentraSite registry. If you want to reset the portlet at a later stage, you identify the portlet using this id.
description	A comment or descriptive information about the portlet.
type	<p>The type of portlet. CentraSite Business UI supports the following portlet types:</p> <ul style="list-style-type: none"> ■ text ■ iframe ■ graphical ■ custom
dataFeed	<i>Required for text/graphical type.</i> Name of a saved search query that feeds in the data to the portlet.
url	<i>Required for iframe type.</i> An arbitrary URL that points to any external HTML page. It renders the HTML content within an HTML IFRAME on the portlet.
icon	<p>An icon to display at the top left corner of the portlet header.</p> <p>The icon is specified as the path to an image file.</p> <p>Prerequisite:</p> <p>The icon must be in PNG format. To ensure proper alignment when it is displayed in the user interface, the icon must be 16 x 16 pixels in size.</p> <p>The icon must reside in the folder <CentraSiteInstall_Directory>\cast\cswbapps\BusinessUI\images\system.</p> <p>The path for the icon should be specified as images/system/icon.png</p>
row, column	The row and column number to place a portlet in the Welcome page.

Parameter	Description
	<p>By default, the portlet is configured to start at level 0. Thus a setting of 0 (zero) refers to the first portlet in the Welcome page.</p> <p>For example, if you want to place a portlet in second row and fourth column, you will need to specify <code>row=1 column=3</code>.</p> <p>However, if at a later state, you attempt to drag and drop a portlet in the Welcome page, it automatically updates the GUI configuration in the repository.</p> <p>Important: To define a portlet of type custom, ensure that you have set the value of <code>row=-1 column=-1</code> in order to dynamically place the custom portlet in your Welcome page.</p>
<code>isVisible</code>	<p>To show or hide the portlet in the Welcome page when you log in to CentraSite Business UI for the first time after product installation. Once you have logged in, you can show or hide the portlet directly from the Welcome page.</p> <p>Possible values: true (default value) – show the portlet or false – hide the portlet.</p>
<code>refreshInterval</code>	<p>The time interval (in seconds) to refresh your portlet content automatically.</p> <p>Note: A value 0 indicates that the portlet content will not be refreshed automatically.</p>
<code>actions</code>	<p>Actions that you want to display in the portlet's configuration.</p> <p>By default, CentraSite supports the following actions:</p> <ul style="list-style-type: none"> ■ Configure ■ Refresh <p>Note: You can specify a comma separated list of actions.</p> <p>Important: When you define a custom action, ensure that you do not specify any of the inherited actions (for example, Expand, Collapse, and Close) explicitly.</p>

Now the Portlet Configuration section looks similar to this.

```
<Portlets>
  <Portlet id="MyFavoritesPortlet"
    description="INMBU_STR_PORTLET_FAVORITES_DESC"
    type="text"
```

```
dataFeed="MyFavorites"
icon="images/system/favorites_16X16.png"
row="0"
column="0"
isVisible="true"
refreshInterval="0"
actions="configure,refresh">
  INMBU_STR_PORTLET_NAME_MY_FAVORITES
</Portlet>
<Portlet id="LastCreatedAssetsPortlet"
  description="INMBU_STR_PORTLET_RECENTLY_CREATED_ASSETS_DESC"
  type="text"
  dataFeed="LastCreatedAssets"
  icon="images/system/recently_created_16X16.png"
  row="1"
  column="0"
  isVisible="true"
  refreshInterval="0"
  actions="configure,refresh">
  INMBU_STR_PORTLET_NAME_LAST_CREATED_ASSETS
</Portlet>
<Portlet id="MyCustomPortlet"
  description="INMBU_STR_PORTLET_CUSTOM_DESC"
  type="text"
  dataFeed="MyFavorites"
  icon="images/system/custom_16X16.png"
  row="1"
  column="1"
  isVisible="true"
  refreshInterval="0"
  actions="configure,refresh">
  INMBU_STR_PORTLET_NAME_MY_CUSTOM_PORTLET
</Portlet>
...
</Portlets>
```

4. Define the actions that you want to render in the portlet's settings.

```
<PortletActions>
  <PortletAction id="configure">
    INMBU_STR_PORTLET_ACTION_EDIT
  </PortletAction>
  <PortletAction id="refresh">
    INMBU_STR_PORTLET_ACTION_REFRESH
  </PortletAction>
</PortletActions>
```

Parameter	Description
id [PortletAction]	A unique identifier for the action. It uniquely distinguishes an action in the CentraSite registry. If you want to reset the action at a later stage, you identify the action using this id.

5. Define the parameters that you want to display in the portlet.

```
<PortletParameters>
  <PortletParameter
```

```

id="MyFavoritesPortletParam" ref="MyFavoritesPortlet"
params="noOfAssets#VSEP#10#PSEP#assetTypeKeys#VSEP#
uddi:cd906138-59f5-4d7f-4f5f-6115adfa8d9c#ASEP#
uddi:cd906138-59f5-4d7f-4f5f-6115adfa8e3d"
attributes="" />
</PortletParameters>

```

Parameter	Description
id [PortletParameter]	A unique identifier for the parameter. It uniquely distinguishes the portlet's parameter in the CentraSite registry. If you want to reset the parameter at a later stage, you identify the parameter using this id.
ref	The specified unique identifier for the portlet.
params	Parameters that you want to display in the portlet's configuration. The parameter values determine which data displays in the portlet.

When you define a parameter, keep the following points in mind:

- A value separator (#VSEP#) is required between the parameter name and the value.

Thus, for example, if you want to define a parameter `noOfAssets` with value `10`, you can write the parameter name-value pair as `noOfAssets#VSEP#10`

- A parameter separator (#PSEP#) is required between the multiple parameters.

Thus, for example, if you want to define two parameters, one parameter `noOfAssets` with value `10`, and another parameter `assetTypeKeys` with value `uddi:cd906138-59f5-4d7f-4f5f-6115adfa8d9c`, you can write the above parameters as

```

params="noOfAssetsToReturn#VSEP#10#PSEP#asset
TypeKeys#VSEP#uddi:cd906138-59f5-4d7f-4f5f-
6115adfa8d9c"

```

- An array separator (#ASEP#) is required between multiple values (that is, an array of values).

Thus, for example, if you want to assign a parameter `assetTypeKeys` with more than one value (`uddi:cd906138-59f5-4d7f-4f5f-6115adfa8d9c`, `uddi:cd906138-59f5-4d7f-4f5f-6115adfa8e3d`), you can write the parameter as

```

assetTypeKeys#VSEP#uddi:cd906138-59f5-4d7f-4f5f-
6115adfa8d9c#ASEP#uddi:cd906138-59f5-4d7f-4f5f-
6115adfa8e3d

```

Parameter	Description
attributes	List of attributes you wanted to display in the portlet's configuration.

6. *For a graphical chart configuration.* If you have opted to show different colors in a graphical chart configuration, specify the colors for each bar, line, or pie plot.

In this case, the line would look like:

```
<Colors>
  <Color id="#233356">OceanBlue</Color>
  <Color id="#038299">LagunaBlue</Color>
</Colors>
```

The `<Colors>` list must have the colors specified in the HEX color code format. The HEX format is a hash (#) followed by 6 numbers or letters. The position of the numbers or letters correlates to the RGB value. For example, `#233356` translates into OceanBlue.

7. Locate the following property statement and specify the required values.

```
<PortletConfigurations
  portletsPerRow="3"
  settingsPopupColumnCount="2"
  helpToken="HELPCENTER_007"
  headerMaxCharacterLength="27">
```

Parameter	Description
PortletsPerRow	The number of portlets to display per row in the Welcome page.
settingsPopupColumnCount	The number of columns to display in the Configure Your Welcome Page dialog.
helpUrl	The URL of the standard Help Center that's delivered with the CentraSite Business UI.
headerMaxCharacterLength	The maximum content length of a portlet's header text. By default, the allowed header text length is 27 characters. If the character in the header text exceeds 27, the text gets automatically truncated.

8. Save and close the configuration file.
9. Restart Software AG Runtime.

Portlet Display Names

The display name for a portlet or an action is the name that is displayed by the CentraSite Business UI. A portlet's or action's display name can either be a localized string fetched from the message database or a plain string.

If you want to specify a localized name, enter the message ID. The message ID (for example, `INMBU_LBL_PORTLET_PENDING_APPROVALS`) must begin with a `INMBU` label. The portlet internally identifies the message ID, and display the localized name, if available in the message database. Else, if you specify a name without `INMBU` label, the portlet displays a plain text name.

Computed Runtime Actions

If you want a runtime workflow to execute a task that is not provided by a built-in runtime action, you can create a custom computed runtime action to perform the work.

The CentraSite Business UI contains a default set of actions, where each action represents a collection of predefined parameters.

Your CentraSite installation contains a sample computed action as an extension in the `<CentraSiteInstall_Directory>/demos` folder. You can adapt the sample computed action to suit your needs and then configure the custom action to display in the policy accordion.

➤ To define a customized computed action

1. Implement the custom runtime action as a GWT extension point. For procedures, see [“Implement Runtime Action as an Extension” on page 95](#).
2. Configure the custom runtime action in **centrasite.xml** configuration file. For procedures, see [“Enable Runtime Action through Configuration” on page 95](#).

Implement Runtime Action as an Extension

Usage	Use this to define a computed runtime action in CentraSite Business UI.
Provided by	<i>BusinessUI</i>
Code	See the sample code.

You can find sample code for defining the custom runtime action as an extension point in the file `CentraSiteBUExtension.gwt.xml`. You can find the file in the `demos` folder in the `<CentraSiteInstall_Directory>`.

Enable Runtime Action through Configuration

After you define the computed action as an extension, enable the computed action configuration so as to display the action in the policy accordion.

Important:

The action parameters defined in the configuration file are editable and cannot be protected.

> To enable runtime action through configuration

1. Open the customization file, `centrasite.xml`, in a rich text editor.

You can find the **centrasite.xml** file on `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\custom\conf`.

2. Locate Policy Action Templates under `<CLL Configuration>`.

The Policy Action Templates snippet looks like this:

```
<PolicyActions>
  <PolicyAction id="" class="" />
</PolicyActions>
```

3. Append the property configuration statement for computed runtime action. For example, `MyComputedRuntimeAction`.

```
<PolicyActions>
  <PolicyAction
    id="uddi:44e3e2de-064c-432f-b67a-8fbca0fb04d6"
    class="com.softwareag.centrasite.bui.
      extension.service.MyComputedRuntimeActionParser" />
</PolicyActions>
```

The input parameters are:

Parameter	Description
id	A unique identifier for the computed action. It uniquely distinguishes an action in the CentraSite registry. If you wish to reconfigure the action at a later stage, you identify the action using this id.
class	A parser implementation for the computed action.

4. Save and close the configuration file.
5. Restart Software AG Runtime.

Installing an Extension

You can use the pluggable architecture to define and install the extensions as part of customizing the Content pages. This section gives the high-level steps of installing an extension to customize the Content pages.

To install an extension

1. Set up a new GWT project in Eclipse. For procedures, see [“Setting up GWT” on page 97](#).
2. Configure the **centrasite.xml** configuration file to adapt to the extension points. For procedures, see [“Configuring an Extension” on page 99](#).

Setting up GWT

- Ensure that you have a recent Eclipse version installed on your machine. Eclipse is available as a download from <http://www.eclipse.org/>. In Eclipse, select **Help > Install New Software** in order to configure usage of GWT plug-in with the Java version currently supported by CentraSite.
- The system requirements can be checked at <http://documentation.softwareag.com/>.

➤ To set up GWT

1. Set the classpath to specify the classpath variables for the new GTW project.
 - a. In Eclipse, select **Window > Preferences > Java > Build Path**
 - b. Expand the **Build Path** node.
 - c. Select the **Classpath Variables**.
 - d. In the Classpath Variables dialog, select **New**. This opens the New Variable Entry dialog box.
 - e. Specify a name for the new classpath variable (CENTRASITE_REDIST_LOCATION) and the location. You can find the `redist` folder in `<CentraSiteInstall_Directory>` (Microsoft Windows) or `/opt/softwareag/CentraSite/redist` (UNIX).
 - f. Click **OK**.
 - g. In the Classpath Variables dialog box, click **OK**.
 - h. Specify a second classpath variable CENTRASITE_RTS_LOCATION. You can find the `rts` folder in `<CentraSiteInstall_Directory>`.

Two new Java classpath variables called CENTRASITE_REDIST_LOCATION and CENTRASITE_RTS_LOCATION have been created due to the previous actions.

- i. Add all of the JAR files contained in the `redist` folder to the classpath.

In addition, if you upgrade to CentraSite 9.7 from a previous version of CentraSite, and you want to use the extension points of the previous version, you must do the following:

- a. Open the MANIFEST.MF file located in the directory `old_<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\META-INF`.
- b. You find the relative path of the jar files, for example, `CentraSiteBUIExtension.jar` and `CentraSiteBUIExtensionCore.jar`, in the `Bundle-ClassPath` section:

```
WEB-INF/lib/CentraSiteBUIExtension.jar,  
WEB-INF/lib/CentraSiteBUIExtensionCore.jar
```

- c. Copy the file path into the `Bundle-ClassPath` section in the MANIFEST.MF file located in the directory `9.7_<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\META-INF`.

Note:

If you have created custom extensions in your previous version of CentraSite, ensure that you include their associated jar files into the META-INF folder.

- d. Add the extension's associated Java archives to the directory `9.7_<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\META-INF`.
- e. Save the modifications.
- f. Restart the Software AG Runtime.

2. Import the GWT project.

- a. Select **File > Import** from the main menu.
- b. On the **Select** page of the Import wizard, select **Existing Projects into Workspace** and click **Next**.
- c. On the Import Projects dialog, choose the **Select root directory** radio button.
- d. Click the **Browse** button that is located beside the input field labeled Select root directory. The Browse for Folder dialog is displayed.
- e. Within the Browse for Folder dialog, navigate to and click on the `CentraSiteBUIExtension` folder of the Software AG Runtime application.

This folder contains the definitions for the extensions that CentraSite supports out of the box. The `CentraSiteBUIExtension` folder resides in the `demos` folder under the CentraSite installation directory.

- f. Click **OK**.
- g. In the Import Projects dialog, click **Finish**.

A new Java project called `CentraSiteBUIExtension` has been created due to the previous actions. This project is now visible in the Package Explorer view in Eclipse. It contains example implementations of the existing extension points for the Business UI. You can modify them to suit your needs. For using your own Widget as Extension Point, specify it

as a return value in the `ExtensionWidgetFactory` class in the `com.softwareag.centrasite.bui.extension.client.factory` package.

- h. Open the file `build.properties`. Type the paths to your `GoogleWebToolkit_HOME`, the Extension target location, the `redist` and the `rts` folder. Now the project can be build using apache ant. Run the `build.xml` script.
- i. To install the extension, use the ant `deploy` target. Open command line and navigate to your project location. Type: `ant deploy`.

Configuring an Extension

We have now created a Java project inside the Business UI web application. The extension functionality for the CentraSite Business UI is contained in the custom configuration file. You can configure this file to adapt to the appropriate extensions as required

➤ To configure an extension

1. Open the customization file, `centrasite.xml`, in a rich text editor.

You can find the **centrasite.xml** file on `<CentraSiteInstall_Directory>\cast\cswbapps\BusinessUI\custom\conf`.

2. Open the file in a text editor, and locate the configuration entry `EXTENSION POINTS`.

```
<Extensions src="">
  <!--centrasitebuiextension/centrasitebuiextension.nocache.js -->
  <ExtensionPointLogin custom="false" />
  <ExtensionPointRegister custom="false" />
  <ExtensionPointBrowseView custom="false" />
  <ExtensionPointBrowseCriteria custom="false" />
  <ExtensionPointSearchWidget custom="false" />
  <ExtensionPointSearch custom="false" />
  <ExtensionPointSearchCriteria custom="false" />
  <ExtensionPointAssetDetails custom="false" />
  <ExtensionPlaces protected="true">
    <ExtensionPlace className=
      "com.softwareag.centrasite.bui.client.place.home.ExtensionPlace"/>
  </ExtensionPlaces>
</Extensions>
```

3. Specify the extensions file path as `CentraSiteBUIExtension/centrasitebuiextension.nocache.js` in order to point to the location of the Java script file.
4. Identify the extension point that you want to configure to the CentraSite Business UI.
5. Enable the extension point by modifying its custom value from false to true. The default value is false.
6. After making the above changes, the extension entry would look like this:

```
<Extensions src="centrasitebuiextension/centrasitebuiextension.nocache.js">
  <ExtensionPointLogin custom="true" />
  <ExtensionPointRegister custom="true" />
  <ExtensionPointBrowseView custom="true" />
  <ExtensionPointBrowseCriteria custom="true" />
  <ExtensionPointSearchWidget custom="true" />
  <ExtensionPointSearch custom="true" />
  <ExtensionPointSearchCriteria custom="true" />
  <ExtensionPointAssetDetails custom="true" />
  <ExtensionPlaces protected="true">
    <ExtensionPlace className=
      "com.softwareag.centrasite.bui.client.place.home.ExtensionPlace" />
  </ExtensionPlaces>
</Extensions>
```

7. Save and close the file.
8. Locate the web.xml configuration file.

You can find the **web.xml** file on `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\WEB-INF`.

9. Open the file in a text editor, and locate the ExtensionServlet entry:

```
<servlet>
  <servlet-name>ExtensionServlet</servlet-name>
  <servlet-class>
    com.softwareag.centrasite.ui.extension.service.ExtensionServiceImpl
  </servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>ExtensionServlet</servlet-name>
  <url-pattern>/centrasitebuiextension/extensionService</url-pattern>
</servlet-mapping>
```

10. Uncomment the ExtensionServlet entry in order to enable the extension configurations that are performed in the previous actions. Similarly, if the extension configuration is already enabled and you want to disable it, simply comment out the ExtensionServlet entry.
11. Save and close the file.
12. Restart Software AG Runtime.

Note:

The extension point's readme file is recommended for reading. The readme file is available in the `<CentraSiteInstall_Directory>\demos\CentraSiteBUIExtension` directory.

Uninstalling an Extension

You can uninstall a single extension or multiple extensions.

Note:

The extension point's readme file is recommended for reading. The readme file is available in the `<CentraSiteInstall_Directory>\demos\CentraSiteBUIExtension`.

➤ To uninstall an extension

1. Open the customization file, `centrasite.xml`, in a rich text editor.

You can find the **centrasite.xml** file on `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\custom\conf`.

2. Open the file in a text editor, and locate the configuration entry for the required extension point.

```
<Extensions src="centrasitebuiextension/centrasitebuiextension.nocache.js">
<ExtensionPointLogin custom="false" />
<ExtensionPointRegister custom="false" />
<ExtensionPointBrowseView custom="false" />
<ExtensionPointBrowseCriteria custom="false" />
<ExtensionPointSearchWidget custom="false" />
<ExtensionPointSearch custom="false" />
<ExtensionPointSearchCriteria custom="false" />
<ExtensionPointAssetDetails custom="false" />
<ExtensionPlaces protected="false">
<ExtensionPlace className=
"com.softwareag.centrasite.bui.client.place.home.ExtensionPlace"/>
</ExtensionPlaces>
</Extensions>
```

3. Remove the configuration entry for the selected extension point.

Alternatively, if you want to uninstall multiple extension points, you can set the extension point's custom value to false.

4. Save and close the file.
5. Restart Software AG Runtime.

Creating a Custom Extension

You can create a custom GWT specific extension to suit your requirements.

➤ To create a custom extension

1. Write a new widget class.
2. Open the file `ExtensionWidgetFactory.java` that resides in the `CentraSiteBUIExtension` folder.
3. Locate the method corresponding to the extension Id whose widget you want to customize.

4. Return the new widget in the corresponding method.
5. Save and close the `ExtensionWidgetFactory.java` file.
6. Edit the custom configuration file to configure the extensions as appropriate. For details on configuring the extension properties, see [“Configuring an Extension” on page 99](#)

Installing CentraSite Business UI Extension

Note:

The `<CentraSiteInstall_Directory>` indicates the installation directory of CentraSite. For example: `C:\SoftwareAG\CentraSite` in Windows or `/opt/softwareag/CentraSite` in Linux.

To install CentraSite Business UI extension

1. Open the `build.properties` file.

You can find the file in the folder `<CentraSiteInstall_Directory>\demos\CentraSiteBUIExtension`.

2. Update the `build.properties` file to include the following information:
 - a. In the property `sag.root.dir`, specify the installation directory of SoftwareAG. For example, `C:\SoftwareAG`.
 - b. In the property `bnd.jar.path`, specify the location of the bnd JAR file.

This property is required for creating JAR files with proper import and export references. The instructions for obtaining the bnd JAR file is provided in the `build.properties` file.

- c. Navigate to the directory where you extracted the Google Web Toolkit (GWT) and include the following Jar files:

- `gwt-dev.jar`
- `gwt-user.jar`
- `validation-api-1.0.0.GA.jar`
- `validation-api-1.0.0.GA-sources.jar`

- d. Set the environment variable `gwt.home` to point to the GWT directory.

3. Run the command `ant`.

The command compiles the `CentraSiteBUIExtension` and create the required files.

The syntax is of the format: `C:\SoftwareAG\CentraSite\demos\CentraSiteBUIExtension>ant`

Deploying CentraSite Business UI Extension

Pre-requisites:

Before you deploy the CentraSite Business UI extension, make sure that you take a backup of the following files:

- `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\WEB-INF\web.xml`
- `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\custom\conf\centrasite.xml`

Note:

The `<CentraSiteInstall_Directory>` indicates the installation directory of CentraSite. For example: `C:\SoftwareAG\CentraSite` in Windows or `/opt/softwareag/CentraSite` in Linux.

Once the CentraSiteBUIExtension is compiled successfully, the BusinessUI extension has to be deployed.

To deploy CentraSite Business UI extension

1. Run the command `ant deploy`.

The command deploys the CentraSiteBUIExtension.

The syntax is of the format: `C:\SoftwareAG\CentraSite\demos\CentraSiteBUIExtension>ant deploy`

2. Open the `web.xml` file.

You can find the file in the folder `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\WEB-INF\`.

3. In the `web.xml` file, uncomment the following property lines:

```
<servlet>
  <servlet-name>ExtensionServlet</servlet-name>
  <servlet-class>com.softwareag.centrasite.bui.
    extension.service.ExtensionServiceImpl</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>ExtensionServlet</servlet-name>
  <url-pattern>/centrasitebuiextension/extensionService</url-pattern>
</servlet-mapping>

<servlet>
  <servlet-name>ActionExecutionServlet</servlet-name>
  <servlet-class>com.softwareag.centrasite.bui.
    extension.core.service.action.ActionExecutorServiceImpl</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>ActionExecutionServlet</servlet-name>
  <url-pattern>/centrasitebuiextension/action</url-pattern>
</servlet-mapping>
```

4. If the `ActionExecutionServlet` is not already available in the `web.xml`, add the following XML fragment below the entry `ExtensionServlet`:

```
<servlet>
  <servlet-name>ActionExecutionServlet</servlet-name>
  <servlet-class>com.softwareag.centrasite.bui.
    extension.core.service.action.ActionExecutorServiceImpl</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>ActionExecutionServlet</servlet-name>
  <url-pattern>/centrasitebuiextension/action</url-pattern>
</servlet-mapping>
```

5. Open the customization file, `centrasite.xml`, in a rich text editor.

You can find the **centrasite.xml** file on `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\custom\conf`.

6. Set the value of the attribute `src` to `centrasitebuiextension/centrasitebuiextension.nocache.js`.

```
src="centrasitebuiextension/centrasitebuiextension.nocache.js"
```

7. In the element `<Extensions>`, set the value of the attribute `custom` to `true` to enable an extension.

The element `<Extensions>` includes all the possible extensions in CentraSite Business UI.

8. To enable the custom actions that are available in the `demos` folder, uncomment the elements `<Actions>` and `</Actions>`.

- If the custom action entries `Action id="RunPolicyAction"` and `Action id="ChangeOwnerAction"` are already available in the **centrasite.xml** file, then uncomment them.
- If the custom action entries `Action id="RunPolicyAction"` and `Action id="ChangeOwnerAction"` are not available in the **centrasite.xml** file, then include the following property statements:

```
<Actions>

  <Action id="RunPolicyAction" inbox="false" bulk="false"
    group="basic"
    icon='images/system/actionbar/execute_policy.png'
    class="com.softwareag.centrasite.demos.bui.
      actions.runpolicy.RunPolicyAction">Execute Policy
  </Action>

  <Action id="ChangeOwnerAction" inbox="false" bulk="true"
    group="basic"
    icon='images/system/actionbar/change_owner.png'
    class="com.softwareag.centrasite.demos.bui.
      actions.changeowner.ChangeOwnerAction">Change Owner
  </Action>
</Actions>
```


9. Save and close the file.
10. Restart Software AG Runtime.

Undeploying CentraSite Business UI Extension

Note:

The `<CentraSiteInstall_Directory>` indicates the installation directory of CentraSite. For example: `C:\SoftwareAG\CentraSite` in Windows or `/opt/softwareag/CentraSite` in Linux.

To undeploy CentraSite Business UI extension

1. Run the command `ant undeploy`.

The command undeploys the CentraSiteBUIExtension.

The syntax is of the format: `C:\SoftwareAG\CentraSite\demos\CentraSiteBUIExtension>ant undeploy`

Note:

You may want to run the command `ant undeploy`, only if the CentraSiteBUIExtension is already be deployed.

2. Open the `web.xml` file, and then undo all changes manually.

You can find the file in the folder `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\WEB-INF\`.

3. Open the customization file, `centrasite.xml`, in a rich text editor.

You can find the **centrasite.xml** file on `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\custom\conf`.

4. Restart Software AG Runtime.

Customizing Design Time Policies

The following policy actions are available as pre-defined policies in CentraSite. Hence, it is not recommended to create another policy for these in the Business UI. To avoid creation of policies with these policy actions, they are hidden from view.

- Default Delete Handler
- Delete Runtime Events and Runtime Metrics
- On Consumer Registration Request Send Email to Owner
- Processing Steps Status
- Set View Permission for Service and Service Related Object to Everyone Group

> To Unhide a Hidden Design Time Policy Action

1. From <SAGDIR>\CentraSite\cast\cswebapps\BusinessUI\custom\conf folder, edit the `centrasite.xml` file.
2. Search for the string **SkippedActions id** and remove the policy action ID from this string to unhide the hidden design time policy action. The string will include the design time policy action IDs as follows:

```
<SkippedActions id="uddi:29361c58-2c0a-11df-b868-b6a0e7c2a55a,uddi:54a474a1-7f3b-418f-833e-a15bc356ff7a,uddi:b1ee9e11-3e25-11de-9eaa-d615f6701759,uddi:4ae84623-98d6-11df-973b-e655342d5efe,uddi:1f7ccebd-9d0a-11de-866d-f6e3a8e65742"/>
```

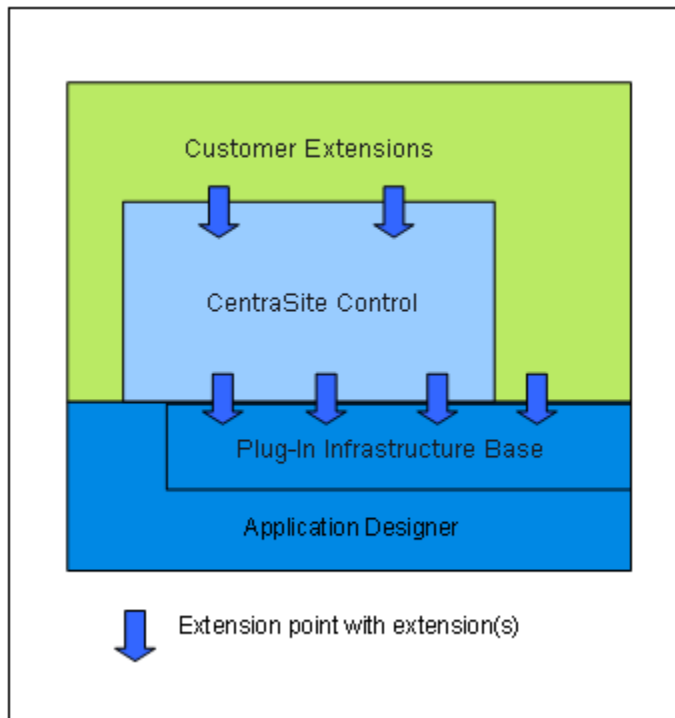
3. Save the XML file.
4. Restart the Software AG Runtime.

Customization of CentraSite Control

CentraSite Control offers a pluggable architecture that allows you to extend the standard graphical interface by adding your own features.

The CentraSite Control user interface is itself a plug-in to a base infrastructure, in other words, the base infrastructure provides extension points where CentraSite Control is plugged in. The base infrastructure is composed of the Application Designer, which provides the basic graphical infrastructure of the GUI and the plug-in infrastructure base, which allows plug-ins to communicate with the Application Designer.

The figure illustrates the plug-in architecture of CentraSite Control.



The plug-in infrastructure is inspired by Eclipse that allows the user interface to be extended by domain-specific or customer-specific functionality.

Plug-ins are implemented as Java classes. The points in the code at which plug-ins can be added are called extension points. CentraSite Control offers extension points that allow you to implement or extend the following features:

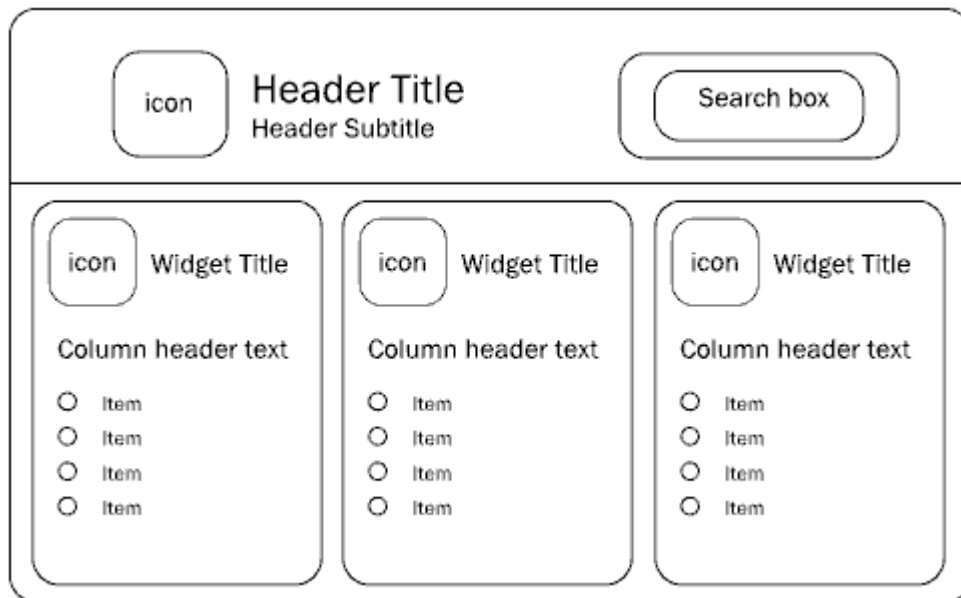
- Provide an alternative login screen.
- Add a topic to the navigation pane within any perspective.
- Support i18N (internationalization) for layouts contributed by a plug-in.
- Add a logo and links to the login dialog.
- Handle the creation and termination of the connection to a backend machine.
- Add a perspective contributing the following components: a toolbar, a logo, one or more topics, and a background screen. A perspective allows you to group topics in the navigation view.
- Add a command to the context menu of a registry object or a repository object.
- Add a property to an object. The property is then visible in detail views and under the **General** tab.
- Add a tab to the detail view of registry objects and repository objects.
- Add a source of notifications.
- Add secondary icons to nodes in the graphical impact analysis.
- Extend the **Summary** tab.

- Replace the standard detail view used as editor for registry and repository objects by an object type specific editor.
- Extend the set of available import commands.
- Extend the search dialog by additional conditions.

A plug-in can itself provide extension points for further plug-ins.

Customizing the Welcome Page

The Welcome page of CentraSite Control can be customized as required. You can change the icons used, colors, text, fonts, and layouts. You can also define links that take you directly to the CentraSite Control pages that are frequently used and links to external web sites. In the Welcome page you can specify the language you use for CentraSite Control and the date format to be used in the various displays. This is the schematic layout for the Welcome page.



In the header section you can change the texts, the fonts, and colors used to display the text in the title and subtitle. You can add an icon adjacent to the title. A search box allows you to perform a keyword search for registry assets and objects whose name or description contains the given keyword. You can hide the search box or change the background color of the search box as part of the page customization. You can change the background color for the whole header section as well.

Below the header section, are the widgets. Each widget contains a title, with an icon adjacent to the text and have a list of entries under the title, each representing some executable action. Typically, an action contains a URL to either a page of your choice within CentraSite Control, or to an external web page that you regularly visit within the context of your work with CentraSite. The Welcome page can contain up to 10 widgets. The widgets are displayed side by side in a single row.

There are several kinds of widget:

- **Single-column widget** - The executable actions are displayed as a table consisting of a single column. Each table cell contains one executable action. Each cell can also have an icon beside it. There is a header text above the table.
- **Multi-column widget** - The executable actions are displayed as a table consisting of two or more columns. Each table cell contains one executable action. Each cell can also have an icon beside it. There is a header text above each column of the table.
- **HTML-style widget** - The contents are freely programmable as HTML code. The HTML statements you use must be valid within the context of an HTML table cell, that is, there is an implicit HTML <td> element enclosing the HTML code you supply.

In general, you can use the CSS stylesheet statements to customize the appearance of text and colors in the Welcome page.

Implementation of Welcome Page

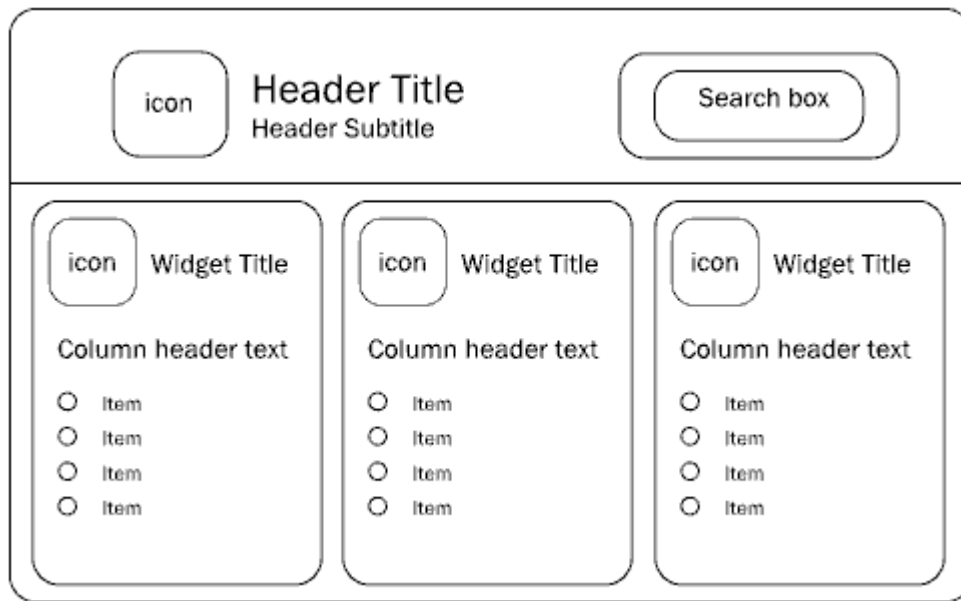
The Welcome page is implemented as a plug-in module within the CentraSite pluggable UI architecture. This means that all the development aspects that are relevant for implementing CentraSite plug-in modules also apply to the Welcome page.

The layout and contents of the Welcome page are implemented as Java code. Each customizable part of the Welcome page requires a corresponding Java class. The Welcome screen can be defined as a combination of the following hierarchical structures:

- The header and body of the Welcome page.
- The widgets in the body of the Welcome page.
- The items in the columns of the widgets.

Screen Component: Welcome Page

The Welcome page is composed of a header and a body. The header contains a title, subtitle, icon, search box, and background image. The body contains one or more widgets. The content and appearance of these components are determined by the Java methods as shown.



The table describes the purpose of these Java methods.

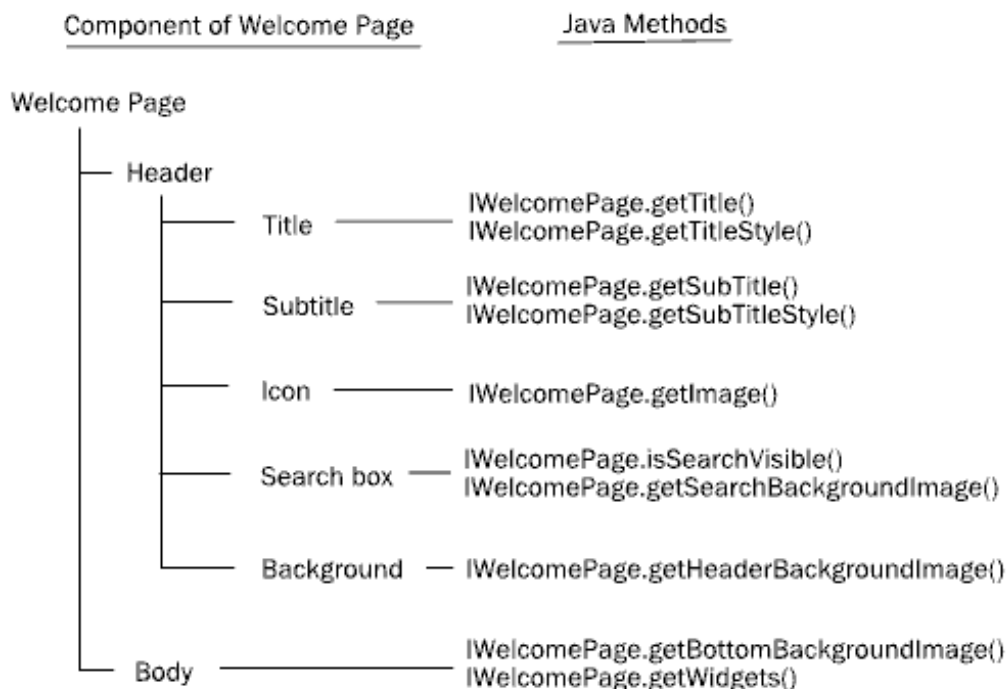
Name of Java interface Java Method		Description
IWelcomePage	getTitle();	Defines the header text to be used.
IWelcomePage	getTitleStyle();	Defines the CSS style information for the header.
IWelcomePage	getSubTitle();	Defines the header subtitle text to be used.
IWelcomePage	getSubTitleStyle();	Defines the CSS style information for the header subtitle.
IWelcomePage	getImage();	Defines the icon to be used in the header.
IWelcomePage	isSearchVisible();	Defines whether the search box in the header part is visible or invisible.
IWelcomePage	getSearchBackgroundImage();	Defines a background image to be used for the search box.
IWelcomePage	getHeaderBackgroundImage();	Defines a background image to be used for the header part.
IWelcomePage	getBottomBackgroundImage();	Defines a background image to be used for the body part.

Name of Java interface	Java Method	Description
IWelcomePage	getWidgets();	Defines the widgets that will be used in the body part.

Screen Component: Widget

The body part of the Welcome page is composed of one or more widgets. A widget can define just HTML code (an HTML-style widget) or can define content and layout, similar to the header part of the Welcome page. The content and layout components are: a background image, a header text, the definition of a single-column table of items, and the definition of a multi-column table of items.

The content and appearance of these components are determined by the Java methods as shown.



The table describes the purpose of these Java methods.

Name of Java interface	Java Method	Description
IWidget	getWidth();	Defines the screen width of the widget.
IHtmlWidget	getHtml();	Defines HTML code for an HTML-style widget.
IColumnWidget	getBackgroundImage();	Defines the background image to be used for a column of a widget.
IColumnWidget	getTitle();	Defines the header text of a column of a widget.

Name of Java interface	Java Method	Description
IColumnWidget	getTitleStyle();	Defines the CSS style for the header text of a column of a widget.
IColumnWidget	getImage();	Defines the background image to be used for the header part of the widget.
ISingleColumnWidget	getSubTitle();	Defines the subtitle text of a single-column widget.
ISingleColumnWidget	getSubTitleStyle();	Defines the CSS style for the subtitle header text of a single-column widget.
ISingleColumnWidget	getItems();	Defines the items contained in a single-column widget.
IMultiColumnWidget	getColumns();	Defines the columns used in a multi-column widget.
IColumn	getSubTitle();	Defines the subtitle text of a column of a multi-column widget.
IColumn	getSubTitleStyle();	Defines the CSS style for the subtitle text of a column of a multi-column widget.
IColumn	getItems();	Defines the items contained in a column of a multi-column widget.

Screen Component: Item

Each widget in the body part of the Welcome page can contain one or more items, arranged in one or more table columns. An item represents an executable action that you can define freely, for example, the action could be the activation of a URL in order to reach a particular page within CentraSite Control or an external web site.

The content and appearance of these components are determined by the Java methods as shown.

<u>Component of Welcome Page</u>	<u>Java Methods</u>
Widget	IWidget.getWidth()
HTML-style widget	IHtmlWidget.getHtml()
Widget with Columns	
Background	IColumnWidget.getBackgroundImage()
Header	IColumnWidget.getTitle() IColumnWidget.getTitleStyle() IColumnWidget.getImage()
Body	
Widget with only one Column	ISingleColumnWidget.getSubTitle() ISingleColumnWidget.getSubTitleStyle() ISingleColumnWidget.getItems()
Widget with one or more Columns	IMultiColumnWidget.getColumns() IColumn.getSubTitle() IColumn.getSubTitleStyle() IColumn.getItems()

The table describes the purpose of these Java methods.

Name of Java interface	Java Method	Description
IItem	getStyle();	Defines the CSS style for the item.
IActionItem	getTitle();	Defines the text to be displayed for the item.
IActionItem	getImage();	Defines the icon to be displayed adjacent to the descriptive text.
ISeparatorItem	getHeight();	Defines the height in pixels of the area that contains the separator image.
ISeparatorItem	getImage();	Defines the image be displayed as the separator item.

Java Methods Not Related to Screen Components

The table lists the Java methods that are not related to a screen component, but are required for the pluggable UI architecture of CentraSite Control.

Name of Java interface	Java Method	Description
(all interfaces)	setLocale();	<p>This informs the widget or item about the CentraSite Control locale that is required to localize texts for the display.</p> <p>This method is called automatically before any other method that might depend on the locale.</p>
(all interfaces)	setActionContext();	<p>This informs the widget or item about the CentraSite Control context that is required for the processing to be done subsequently by the execute() method.</p> <p>This method is called automatically before any other method that might depend on the action context.</p>
IWidget	invalidate();	This sets the status that the display of the current item must be refreshed (true) or does not need to be refreshed (false).
IWidget	isInvalidated();	This returns whether or not the display of the item needs to be refreshed.
IItem	getWidget();	This method gets the widget to which the current item belongs.
IItem	setWidget();	This method sets the widget to which the current item belongs.
IActionItem	execute();	This activates the action to be performed when you click on the current item.

Java Interface Hierarchy

The interface hierarchy is as follows:

```
IWelcomePage

IWidget
  IColumnWidget
    IMultiColumnWidget
    ISingleColumnWidget
  IHtmlWidget

IColumn

IItem
  IActionItem
  ISeparatorItem
```

Installing the Customized Welcome Page

The Welcome page is implemented as a CentraSite Control extension point in the context of CentraSite's pluggable UI architecture. To install the customized Welcome page, you have to modify CentraSite Control's pluggable UI configuration in the Software AG Runtime environment.

1. Stop Software AG Runtime.
2. Update the `plugin.xml` configuration file.

The standard `plugin.xml` configuration file delivered with the CentraSite kit contains all the names of the CentraSite Control extension points, including the extension point for the Welcome page. You must update this file to contain the definition of the customized Welcome page. The configuration file is located in the folder `<RuntimeDir>\workspace\webapps\PluggableUI\CentraSiteControl`.

There are two elements in the standard `plugin.xml` file that refer to the Welcome page. The first part defines the name of the extension point to be used for the Welcome page.

```
<extension-point id="welcomePage">
</extension-point>
```

The second part defines the Java class that implements the Welcome page.

```
<!-- Welcome Page -->
<extension
  point="com.centrasite.control.welcomePage"
  id="welcomePage"
  class="com.centrasite.control.ext.welcome.standard.WelcomePage">
</extension>
```

The `point` attribute of the `extension` element in the second part must match the name given by the `id` attribute of the `plugin` element (usually in the first line in the `plugin.xml` file) concatenated with a dot and the `id` attribute of the `extension-point` element in the first part. For example, if the `id` attribute of the `plugin` element is `com.centrasite.control` and the `id` attribute of the `extension-point` element is `welcomePage`, then the value of `point` attribute of the `extension` element must be `com.centrasite.control.welcomePage`. The Java class identified by the `class` attribute of the `extension` element must implement the interface `IWelcomePage`. To use the customized Welcome page instead of the standard Welcome page, set the `class` attribute to the customized Java class that implements the interface `IWelcomePage`. The changes that you make in `plugin.xml` take effect the next time Software AG Runtime is started.

Note:

Instead of overwriting the standard element in `plugin.xml`, you may want to retain a copy of the original element and comment it out. This helps when you want to revert to the original Welcome page.

3. Deploy the new Java classes to the PluggableUI environment.

You have to copy the Java classes for your customized Welcome page to the CentraSite Control location in Software AG Runtime. There are two ways of doing this:

- Create a jar file containing the class files for the customized Welcome page, and copy the jar file to the `CentraSiteControl\lib` folder in Software AG Runtime.
- Copy the class files to the `CentraSiteControl\classes` folder and its subfolders, according to the naming convention of the Java package that contains the classes. For example, if your package name is `com.centrasite.control.ext.welcome.sample`, then copy the classes to the `CentraSiteControl\classes\com\centrasite\control\ext\welcome\sample` folder.

You can also combine these methods and define some classes using a jar file in the `lib` folder and some classes as class files in the appropriate subfolder of the `classes` folder. If you have defined a class in both `lib` and a subfolder of `classes`, the class in the `CentraSiteControl\classes` subfolder will be used. If you have defined new icons for the customized Welcome page, you have to copy the icons to the appropriate location under the `CentraSiteControl` folder. If, for example, your code contains the definition `public String getImage() { return "images/my_welcome_icon.png"; }`, ensure that the icon `my_welcome_icon.png` is copied to `CentraSiteControl\images`.

4. Start Software AG Runtime.

The changes you have made should now be visible when you view the Welcome Page.

Sample Customized Welcome Page

This section describes the customized Welcome page that is provided as a demo in the product distribution.

Location of Demo Files

The required files for the demo are located in the folder `demos\WelcomePage` under the `CentraSite` installation location. The files are:

- The Java source files. These are located in the subfolder `src`.
- Icons to be displayed in the Welcome page. These are located in the subfolder `resources`.
- Updates for the Software AG Runtime configuration. These are in the file `resources\plugin.xml`.
- Eclipse project files `.classpath` and `.project`.
- Apache Ant files `build.properties` and `build.xml` for building the files that are deployed to Software AG Runtime.

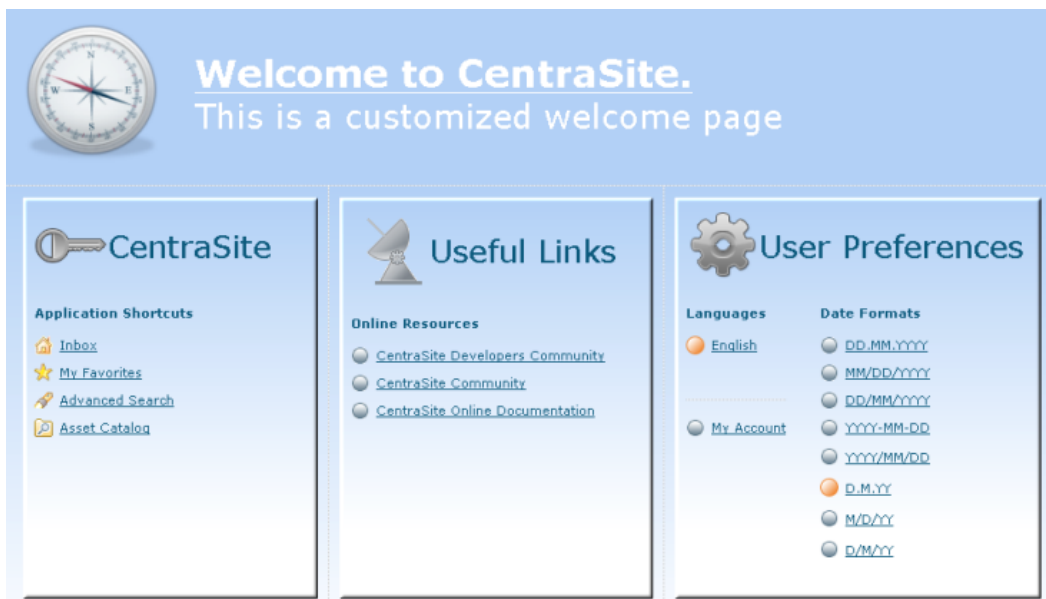
Differences Between the Standard Welcome Page and the Demo Welcome Page

The figures display the differences between the standard welcome page and the demo welcome page that is customized as required.

The standard welcome page appears as follows:



The demo welcome page used as an example in this section appears as follows:



The main changes between the standard welcome page and the customized welcome page that Software AG supplies as a demo are:

- The text in the title of the header section has changed. Also the color of this text has changed.
- The background color in the customized welcome page has changed.
- The large icons in the titles of the header part and of the widgets have changed.
- The small icons in the CentraSite widget have changed.
- The widgets have 3-D effect shadowed borders.

- The search box in the header section has been removed.

Implementation of Welcome Page Layout

This section lists the layout possibilities of the welcome page and specifies the Java methods where the layout is defined.

Note:

If any background image that is defined for an area of display is not as wide as the area, the image is repeated horizontally until the whole width of the area is covered.

Header Area

Layout component	Source file	Java Method
Icon	WelcomePage.java	getImage();
Background image	WelcomePage.java	getHeaderBackgroundImage();
Title text	WelcomePage.java	getTitle();
CSS style of title text	WelcomePage.java	getTitleStyle();
Subtitle text	WelcomePage.java	getSubTitle();
CSS style of subtitle text	WelcomePage.java	getSubTitleStyle();
Background image of the Search box	WelcomePage.java	getSearchBackgroundImage();
Make the Search box visible/invisible	WelcomePage.java	isSearchVisible();

Separator Between Header Part and Widget Part

Layout component	Source file	Java Method
Image	SeparatorItem.java	getImage();
Height in pixels	SeparatorItem.java	getHeight();

Widget CentraSite

Layout component	Source file	Java Method
Width of widget	CentraSiteWidget.java	getWidth();
Title text	CentraSiteWidget.java	getTitle();
CSS style of title text	CentraSiteWidget.java	getTitleStyle();
Subtitle text	CentraSiteWidget.java	getSubTitle();

Layout component	Source file	Java Method
CSS style of subtitle text	CentraSiteWidget.java	getSubTitleStyle();
Header icon	CentraSiteWidget.java	getImage();
Background image	CentraSiteWidget.java	getBackgroundImage();
CentraSite widget: define the items to be included in the bullet list	CentraSiteWidget.java	getItems();
Icon for item Asset Catalog	KeywordSearchItem.java	getImage();
Text for item Asset Catalog	KeywordSearchItem.java	getTitle();
Icon for item Advanced Search	AdvancedSearchItem.java	getImage();
Text for item Advanced Search	AdvancedSearchItem.java	getTitle();
Icon for item Inbox	InboxItem.java	getImage();
Text for item Inbox	InboxItem.java	getTitle();
Icon for item My Favorites	MyFavoriteItem.java	getImage();
Text for item My Favorites	MyFavoriteItem.java	getTitle();

Widget Useful Links

Layout component	Source file	Java Method
Header icon	UsefulLinksWidget.java	getImage();
Width of widget	UsefulLinksWidget.java	getWidth();
Title text	UsefulLinksWidget.java	getTitle();
CSS style of title text	UsefulLinksWidget.java	getTitleStyle();
Subtitle text	UsefulLinksWidget.java	getSubTitle();
CSS style of subtitle text	UsefulLinksWidget.java	getSubTitleStyle();
Background image	UsefulLinksWidget.java	getBackgroundImage();
Text for item CentraSite Developers Community	DeveloperCommunityItem.java	getTitle();
URL for item CentraSite Developers Community (See note below)	DeveloperCommunityItem.java	execute();

Layout component	Source file	Java Method
Text for item CentraSite Community	CentraSiteCommunityItem.java	getTitle();
URL for item CentraSite Community (See note below)	CentraSiteCommunityItem.java	execute();
Text for item CentraSite Online Documentation	OnlineDocumentationItem.java	getTitle();
URL for item CentraSite Online Documentation (See note below)	OnlineDocumentationItem.java	execute();
Define the items to be included in the bullet list	UsefulLinksWidget.java	getItems();

Note:

For the URLs for items CentraSite Developers Community, CentraSite Community, and CentraSite Online Documentation, the creation of a hyperlink that opens a new browser page is implemented by a call of the `openPageInNewWindow` method of the `getDisplayAdapter()` class that is available in the `CentraSiteControlUI.jar` file in Software AG Runtime.

Widget User Preferences

Layout component	Source file	Java Method
Header icon	UserPreferencesWidget.java	getImage();
Width of widget	UserPreferencesWidget.java	getWidth();
Title text	UserPreferencesWidget.java	getTitle();
CSS style of title text	UserPreferencesWidget.java	getTitleStyle();
Background image	UserPreferencesWidget.java	getBackgroundImage();
Text of the Languages subtitle	LanguagesColumn.java	getSubTitle();
CSS style of the Languages subtitle	LanguagesColumn.java	getSubTitleStyle();
Width of Languages column in pixels	LanguagesColumn.java	getWidth();
Text of the Date Formats subtitle	DateFormatsColumn.java	getSubTitle();
CSS style of the Date Formats subtitle	DateFormatsColumn.java	getSubTitleStyle();

Layout component	Source file	Java Method
Width of Date Formats column in pixels	DateFormatsColumn.java	getWidth();
Languages column: define the items to be included in the bullet list	LanguagesColumn.java	getItems();
Date Formats column: define the items to be included in the bullet list	DateFormatsColumn.java	getItems();

Default Settings for Widgets

Layout component	Source file	Constant
Widgets: default "blue circle" icon to mark individual entries in a widget	WelcomePage.java	BLUE_CIRCLE_ICON
Widgets: default "orange circle" icon to mark individual entries in a widget	WelcomePage.java	ORANGE_CIRCLE_ICON
Widgets: default CSS style of the title text of a widget	WelcomePage.java	WIDGET_TITLE_STYLE
Widgets: default CSS style of the subtitle text of a widget	WelcomePage.java	WIDGET_SUBTITLE_STYLE
Widgets: default CSS style of the text for each item of a widget	WelcomePage.java	ACTION_ITEM_STYLE

Special Programming Techniques

This section summarizes some of the techniques you may find useful when creating a customized Welcome page. You can find code examples of the techniques in the `demos\WelcomePage` folder.

Technique	Code Example in <code>demos\WelcomePage</code> folder
Activate the Advanced Search page.	AdvancedSearchItem.java
Activate the Keyword Search page.	KeywordSearchItem.java
Start the My Account dialog.	UserPreferencesItem.java
Start the Add Asset dialog.	CreateAssetItem.java

Technique	Code Example in demos\WelcomePage folder
Start the Import dialog.	ImportWsdlFileItem.java
Activate My CentraSite and show Assets I Provide.	MyFavoriteItem.java
Open the external website http://communities.softwareag.com/centrasite .	CentraSiteCommunityItem.java
Open the external website http://www.centrasite.com .	DeveloperCommunityItem.java
Open the external website http://documentation.softwareag.com/default.htm .	OnlineDocumentationItem.java
Create an empty line in a widget.	EmptyItem.java
Create a dotted dividing line.	SeparatorItem.java
Create a column (list) with all available date formats.	DateFormatsColumn.java
Select a specific date format from a list.	DateFormatItem.java
Create a column (list) with all available languages.	LanguagesColumn.java
Select a specific language from a list.	LanguageItem.java

Implementing the Sample Customized Welcome Page

1. Implement the demo as an Eclipse Java project.

a. Start Eclipse.

If you want to use Eclipse as your development environment for updating the Java sources of the customized welcome page, the demos\WelcomePage folder contains the Eclipse project files `.classpath` and `.project`. You can use these files to create an Eclipse Java project for managing your Java sources.

b. Select **File > New > Project > Java Project**.

c. Select **Create project from existing source**.

d. Specify the path `demos\WelcomePage` as the location of the existing project files.

When you build the project in Eclipse (using for example **Project > Build Project**), there should be no errors reported.

2. Build the deployment files for Software AG Runtime.

- a. Build the jar file by using Apache Ant with the build file `build.xml` provided in the `demos\WelcomePage` folder.

To deploy the demo Welcome page to Software AG Runtime, you have to create a jar file containing the Java classes of the Java sources, then copy the jar file and any required graphic icons to the Software AG Runtime environment.

The file `build.xml` uses a properties file `build.properties` to define some customer-specific files names and folder locations. The build file, `build.xml` also builds a zip file that contains the jar file and all required graphical icons. To deploy the demo welcome page, you can extract the contents of the zip file directly into your Software AG Runtime location.

The file `build.properties` contains the following properties that you should tailor to your working environment before you run `build.xml`.

Property	Description
<code>projectName</code>	<p>This is the name that is used for the jar file and zip file that are created by the Ant task.</p> <p>The jar file is copied to the <code>CentraSite\lib</code> folder in the Software AG Runtime environment, so select a name that is easily distinguishes the jar file from other jar files at the Software AG Runtime location.</p>
<code>pluggableLocation</code>	<p>This is the location of the <code>webapps/PluggableUI</code> folder in the Software AG Runtime environment. In a Windows environment, you should use forward slash instead of backward slash in the path name.</p>
<code>centraSiteLocation</code>	<p>This is the path where your CentraSite installation is located. In a Windows environment, you should use forward slash instead of backward slash in the path name.</p>

3. Build the deployment files using Eclipse or from Command Line.

The `build.xml` file contains the definition of the tasks to be performed by Ant. The tasks defined in the delivered demo version are:

- Compile the Java sources that are located in the folder `src` and store the Java classes in the folder `classes`.
- Create a jar file containing all of the class files, and store the jar file in the folder `lib`.
- Create a zip file that contains the jar file and all icons associated with the customized welcome page and store the zip file in the folder `lib`.

The `build.xml` file is an XML file that contains element definitions such as:

```
<zipfileset dir="resources" prefix="images"> <include name="*.png" /> </zipfileset>
```

In such cases, the `dir` attribute indicates the name of the folder in the build environment where Ant can locate the required files, and the `prefix` attribute indicates the folder in the Software AG Runtime environment where the files are copied to. In the extract shown above, Ant searches for all PNG graphics files ("*.png") in the `resources` folder in the build environment and adds them to the zip file so that they can be extracted into the `images` folder in the Software AG Runtime environment.

Whether you build the deployment files using Eclipse or the command line, the Ant tasks defined in the `build.xml` are processed. Ant builds a new jar file `demos\WelcomePage\lib\SagBlueWelcomePage.jar`, containing all of the Java classes required for the Software AG Runtime environment. It also builds a zip file `demos\WelcomePage\lib\SagBlueWelcomePage.zip`, containing the jar file and all required PNG graphics. The name `SagBlueWelcomePage` comes from the definition of the property `projectName` in the file `build.properties`.

■ Building the Deployment Files Using Eclipse

1. In Eclipse, select the `build.xml` file in the Package Explorer view.
2. In the context menu, click **Run As > Ant Build....**
3. Ensure that the options are set for `Clear Environment`, `Compile Sources`, `Create JAR file`, `Create ZIP file`.
4. Click **Run**.

■ Building the Deployment Files from the Command Line

1. Open a command prompt window.
2. Go to the `demos\WelcomePage` folder.
3. Type the command `ant clean`.
4. Type the command `ant`.

4. Deploying the Demo to Software AG Runtime

- a. Stop Software AG Runtime.
- b. Do one of the following alternatives:

- Unzip the zip file created by the Ant build into `<RuntimeDir>\workspace\webapps\PluggableUI\CentraSiteControl` directory.

This copies the jar file created by Ant into the folder `<RuntimeDir>\workspace\webapps\PluggableUI\CentraSiteControl\lib` and the PNG files into the folder `<RuntimeDir>\workspace\webapps\PluggableUI\CentraSiteControl\images`.

- As an alternative to using the zip file, you can just copy the jar file from the Ant build into `<RuntimeDir>\workspace\webapps\PluggableUI\CentraSiteControl\lib` and the PNG files into `<RuntimeDir>\workspace\webapps\PluggableUI\CentraSiteControl\images`.

- c. Update the `plugin.xml` file in the Software AG Runtime environment to point to the Java classes of the customized Welcome page, as described in [“Implementing the Sample Customized Welcome Page” on page 122](#). The file `plugin.xml` in the folder `demos\WelcomePage\resources` contains the elements that must be updated in the `plugin.xml` file for Software AG Runtime.

Copy the entries manually from `demos\WelcomePage\resources\plugin.xml` to the `plugin.xml` file under Software AG Runtime. Remember to comment out the original entries for the standard Welcome page when you copy in the new entries.

- d. Restart Software AG Runtime.
5. Display the demo Welcome page.

After you have deployed the demo to the Software AG Runtime environment and restarted Software AG Runtime, the demo Welcome page is visible when you start CentraSite Control.

Customizing Content Pages

To customize Content pages, you can use extension points for the CentraSite Control and implement Java classes and methods to plug in to the extension points. An extension point is characterized by the following properties:

- An ID by which it can be referenced.
- An interface to be implemented by plug-ins. In most cases there is also an abstract base class available that implements the interface. It is recommended to extend this class for your own extensions.
- Names of properties to be provided by a plug-in.
- Optionally, it may be related / compared to a corresponding extension point offered in an Eclipse environment.

An extension point provides the name of a class that implements the interface and property values. In general, if there is an abstract base class, its usage is strongly encouraged.

Extension Points

I18N for Layouts

Usage	Use this when the layout of a plug-in needs to be localized.
Attributes	<ul style="list-style-type: none"> ■ <code>point="com.softwareag.cis.plugin.i18n"</code> ■ <code>id</code> ■ <code>class</code>

- project (name of the plug-in directory)
- prefix (as used by messages)
- file (name of the property file to be used)

Interface	I18NHandler
Standard class	Common18NHandler
Processing	Class I18NManager inside PluggableUI handles this extension point. If an I18NMessage or a text ID in a layout definition (as created using the Application Designer) refers to a source ID that starts with the given prefix, the I18NManager attempts to resolve this reference using the given property file. In the case of a text ID, the corresponding layout must be part of the plug-in whose directory is indicated by the project attribute.
Provided by	PluggableUI
Example	<pre><extension point="com.softwareag.cis.plugin.i18n" id="CentraSiteControl" class="com.softwareag.cis.plugin.ext.Common18NHandler" project="CentraSiteControl" prefix="INMCS" file="com.centrasite.control.adapters.util.INMMessages"> </extension></pre>

Parameters for Plug-ins

Usage	Use this to get parameters for a plug-in.
Attributes	<ul style="list-style-type: none">■ point="com.softwareag.cis.plugin.parameter"■ id■ value
Interface	No interface to be implemented.
Processing	Use the plug-in call <code>ApplicationContext.getParameter()</code> to obtain value.
Provided by	PluggableUI
Example	<pre><extension point="com.softwareag.cis.plugin.parameter" id="welcomePageDefault" value="true"> </extension></pre>

ConnectionHandler - Logon and Logoff or Exit

Usage	Use at the start or end of a session of CentraSite Control.
--------------	---

Attributes	<ul style="list-style-type: none"> ■ point="com.softwareag.cis.plugin.connectionHandler" ■ id ■ value
Interface	<p>ConnectionHandler</p> <ul style="list-style-type: none"> ■ void init (CommonAdapter ca) ■ void connect (Credentials c, CommonAdapter ca) throws Exception ■ void notifyConnected (CommonAdapter ca) ■ boolean isConnected() ■ void prepareDisconnect (CommonAdapter ca) throws Exception ■ void disconnect (CommonAdapter ca);
Processing	<ul style="list-style-type: none"> ■ Logon: <ul style="list-style-type: none"> ■ Obtain credentials from the login screen ■ Call connect(credentials) for each extension ■ If an exception occurs: <ul style="list-style-type: none"> ■ Show a popup with the exception ■ Disconnect each extension which is already connected ■ Restart ■ If all successful: start the workplace ■ Logoff: <ul style="list-style-type: none"> ■ Call prepareDisconnect() for each extension ■ If an exception occurs: <ul style="list-style-type: none"> ■ Show a popup with the exception ■ Done ■ Disconnect each extension which is already connected by calling the disconnect() method
Provided by	PluggableUI
Example	<pre><extension point= "com.softwareag.cis.plugin.connectionHandler" id="login" class="com.centrasite.control.ext. CentraSiteConnectionHandler"> </extension></pre>

Perspectives

Perspectives allow certain predefined screen layouts to be stored. When several perspectives are defined, it is possible to switch from one to the other easily. The **Perspective** button is visible if more than one perspective is available. When you click the button, a popup dialog appears, which allows you to select the required perspective. The perspective can be switched either by selecting one or more rows (perspectives) and clicking **OK** or by double-clicking on a single row.

Multiple perspectives will be represented in a way that the union of the corresponding topics is displayed on the right-hand side. The header changes depending on the perspective to which the currently selected topic belongs.

The following features are provided for perspectives:

- A fixed set of perspectives as configured through extensions. You can switch a perspective using the **Select Perspective** dialog.
- A fixed set of topics per perspective. The association between topics and the corresponding perspective is established through the plug-in configuration file. Each declaration of a topic extension must contain a reference to the ID of the associated perspective extension.
- A perspective may contribute the following components:
 - A name and an icon being used to represent the perspective in the **Select Perspective** dialog.
 - An `ICONLISTInfo` object used to create a toolbar in the header frame. This can be suppressed if the perspective's `supportsViews()` method returns `false`.
 - The label and valid values for the **View** list box. This can be suppressed.
 - A tailored layout to be used as the workplace background. This is only used if the perspective is used as the initial perspective. For more information, see [“Setting Preferred Plug-in and Order of Plug-ins” on page 142](#).

Usage	Each plug-in may contribute a perspective to contain its own topics or the topics of other plug-ins
Attributes	<ul style="list-style-type: none">■ <code>point="com.softwareag.cis.plugin.perspective"</code>■ <code>id</code>■ <code>class</code>
Interface	<ul style="list-style-type: none">■ <code>Perspective</code><ul style="list-style-type: none">■ <code>String getTitle()</code> (used in dynamically generated Select Perspective dialog)■ <code>String getImageURL()</code>

(used to represent a perspective by an icon in the **Select Perspective** dialog)

- **Toolbar:**
 - `ICONLISTInfo getToolbar()`
- **Logo**
 - `String getLogoImageUrl ()`
(used for header frame)
- **Handling of the **View** listbox:**
 - `String getViewLabel()`
 - `String[] getViewValues()`
 - `String getView()`
(returns the currently selected view)
 - `void setView(String view)`
(called when the user changes the view selection)
- **Default layout used for perspective background**
 - `String getWorkplaceDefaultLayout();`
(used for background of workplace if no activity is opened)

Abstract base class `AbstractPerspective`

Provided by `PluggableUI`

Example `(CentraSite Control/plugin.xml)`

```
<extension point="com.softwareag.cis.plugin.perspective"
  id="controlPerspective" <----+
  class="com.centrasite.control.ext.ControlPerspective"> |
</extension> |
<extension point="com.softwareag.cis.plugin.topic" |
  id="registry" |
  perspective="com.centrasite.control.controlPerspective" ----+
  class="com.centrasite.control.ext.ImportantTypesTopic">
</extension>
```

If a perspective is selected for display, all topics belonging to that perspective become visible. If one of these perspectives is selected in the navigation pane, the content of the header frame is adjusted with respect to the toolbar, the View listbox, and the visible logo.

Topic

Usage	Add a topic in the navigation view.
Attributes	<ul style="list-style-type: none">■ <code>point="com.softwareag.cis.plugin.topic"</code>■ <code>id</code>■ <code>perspective</code>■ <code>class</code>
Interface	Topic <ul style="list-style-type: none">■ <code>String getImageURL()</code>■ <code>boolean isVisible()</code> (used when switching views)
Abstract base class AbstractTopic	
Processing	<ul style="list-style-type: none">■ When starting the user interface, a topic is added to the active perspective for each known extension that refers to the perspective.■ The first topic is selected.■ When switching to a different topic, replace the content of the HEADER frame according to the data provided by the corresponding perspective.
Provided by	PluggableUI
Example	<pre><extension point="com.softwareag.cis.plugin.topic" id="registry" perspective="com.centrasite.control.perspective" class="com.centrasite.control.ext.ImportantTypesTopic"> </extension></pre>

Command for Item

Usage	Add a command to a menu.
Attributes	<ul style="list-style-type: none">■ <code>point="com.centrasite.control.itemCommand"</code>■ <code>id</code> (default: name of implementing class)■ <code>class</code>
Interface	ExtensionCommand

- boolean appliesTo (Item)
- String getName()
- String getImageURL()
- int getCategory()
(used for grouping of commands)
- abstract void execute(ActionContext actionContext)

Abstract base class AbstractExtensionCommand

- Processing**
- When a list of commands for a menu item is retrieved (for example, for context menu or toolbar), the following steps are performed for each known extension:
 - Create an instance of class and invoke appliesTo(Item).
 - If true is returned, the command is added to the list.

Provided by CentraSite Control

Example

```
<extension point="com.centrasite.control.itemCommand"
  id="test"
  class="com.centrasite.control.extpt.junit.
  DisplayRegObjKeyCommand">
</extension>
```

Bulk Command for Items

Usage Add a command to a menu in which bulk actions are permitted.

- Attributes**
- point="com.centrasite.control.itemBulkCommand"
 - id (default: name of implementing class)
 - class

Interface ExtensionCommand

- boolean appliesTo (Item)
 - String getName()
 - String getImageURL()
 - int getCategory()
(used for grouping of commands)
 - abstract void execute(ActionContext actionContext)
-

Abstract base class AbstractExtensionCommand

- | | |
|-------------------|--|
| Processing | <ul style="list-style-type: none">■ When a list of commands for a menu item is retrieved (for example, for context menu or toolbar), the following steps are performed for each known extension:<ul style="list-style-type: none">■ Create an instance of class and invoke <code>appliesTo(Item)</code>.■ If <code>true</code> is returned, the command is added to the list. |
|-------------------|--|
-

Provided by	CentraSite Control
--------------------	--------------------

Example	<pre><extension point="com.centrasite.control.itemBulkCommand" id="test" class="com.centrasite.control.extpt.junit. DisplayRegObjKeyCommand"> </extension></pre>
----------------	--

Add Property

Usage	Add a property to a registry object.
--------------	--------------------------------------

- | | |
|-------------------|---|
| Attributes | <ul style="list-style-type: none">■ <code>point="com.centrasite.control.registryObjectProperty"</code>■ <code>id</code>■ <code>class</code>■ (boolean) <code>visible by default</code> |
|-------------------|---|
-

- | | |
|------------------|--|
| Interface | <p>ExtensionPropertyAccessor</p> <ul style="list-style-type: none">■ <code>boolean appliesTo (String objectTypeQName, Connector con)</code>■ <code>String getDisplayName(Locale locale)</code>■ <code>String getDescription(Locale locale)</code>■ <code>String getInternalName()</code>■ <code>boolean getVisibleByDefault()</code>■ <code>String getValue(Item item) throws Exception</code>■ <code>void setValue(Item item, String value) throws Exception</code> |
|------------------|--|
-

Abstract base class AbstractPropertyAccessor (must be explicitly implemented)

- | | |
|-------------------|---|
| Processing | <ul style="list-style-type: none">■ When opening a report, all extensions are checked whether they want to contribute.■ The corresponding accessors are added to the report. |
|-------------------|---|
-

Provided by	CentraSite Control
--------------------	--------------------

Example

```
<extension
  point="com.centrasite.control.registryObjectProperty"
  id="test"
  class="com.centrasite.control.extpt.junit.
  LastModifiedPropertyAccessor">
</extension>
```

Note:

Additional columns may also show up in upper table of the **General** tab.

Tab in Detail View

Usage	Add a tab in the detail view of an object.
Attributes	<ul style="list-style-type: none"> ■ point="com.centrasite.control.detailViewTab" ■ id ■ class
Interface	<p>DetailViewTab</p> <ul style="list-style-type: none"> ■ String getTitle() ■ String getImageURL() ■ String getLayout() ■ void initAdapterFor (Item, DetailViewTabAdapter) ■ protected String getAdapterClass(); ■ boolean appliesTo (Item) <p>(If this is returned, the tab will be displayed for the corresponding Item if isVisible() returns true as well, otherwise the tab will not be displayed)</p> <ul style="list-style-type: none"> ■ void setDetailsTabContext (DetailTabContext) ■ boolean isVisible(Item)
Abstract base class	AbstractDetailViewTab
Processing	<ul style="list-style-type: none"> ■ If the detail view for an Item is opened, it is checked for each known extension. <ul style="list-style-type: none"> ■ Create an instance of class and invoke appliesTo(Item). If true is returned, getLayout() is invoked and the layout is added as a tab. The respective adapter is created implicitly by the Application Designer when processing the layout. ■ The title of the tab is set with the result from calling getTitle().

- Currently, items on tabs are not supported. Hence, the result from `getImageURL()` is ignored.

Provided by	CentraSite Control
Example	<pre><extension point="com.centrasite.control.detailViewTab" id="lifecycle" class="com.centrasite.control.lifecycle.LifeCycleDetails"> </extension></pre>

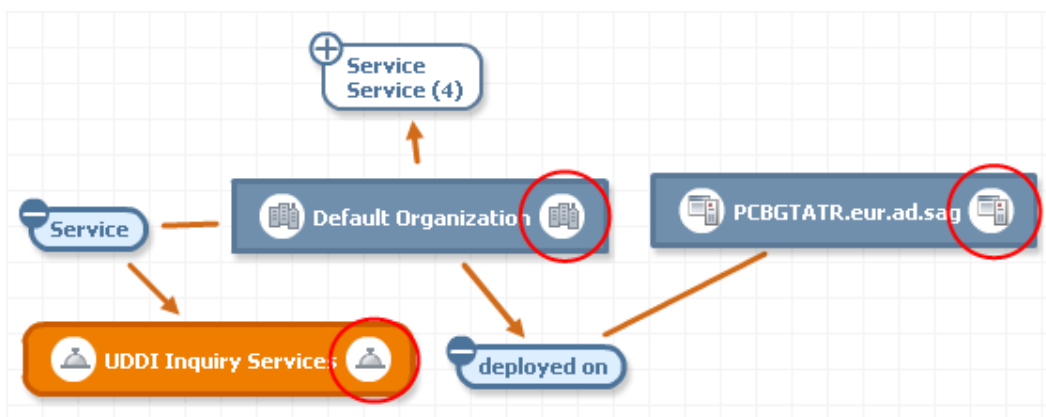
Add Source of Notification

Usage	Add a source of a notification.
Attributes	<ul style="list-style-type: none">■ <code>point="com.centrasite.control.addRowToMyNotifications"</code>■ <code>id</code> (default: name of implementing class)■ <code>class</code>
Interface	<p>ReportExtensionItemsProvider</p> <ul style="list-style-type: none">■ <code>Collection getItems()</code> throws <code>Exception</code>;■ <code>void setConnector(Connector connector);</code>■ <code>boolean isContributedItem(Item item);</code>■ <code>String getChangedImageUrl (Item item);</code>
Abstract base class	AbstractReportExtensionItemsProvider
Processing	<ul style="list-style-type: none">■ The extension is initialized using the <code>setConnector()</code> method.■ Obtain all items to be added to the list of items with pending notification using the <code>getItems()</code> method.■ <code>isContributedItem()</code> can be used to check whether this extension has contributed the given item through <code>getItems()</code>.■ <code>getChangedImageUrl()</code> is used to control the icon representing the reason for the notification.
Provided by	CentraSite Control
Example	<pre><extension point="com.centrasite.control.addRowToMyNotifications" id="MyNotificationsApprovalItemsProvider" class="com.softwareag.centrasite.control.lms.ext. MyNotificationsApprovalItemsProvider"> </extension></pre>

Impact Analysis: Node Decorator

Usage	Change the visual representation of registry objects
Attributes	<ul style="list-style-type: none">point="com.centrasite.control.assocNavigatorNodeDecorator"id (default: name of implementing class)class
Interface	NodeDecorator <ul style="list-style-type: none">String getImageURL (Item)
Abstract base class	(none)
Processing	<p>If the item is to be rendered in Impact Analysis, check all known extensions to determine whether they contribute to the item's visualization.</p> <ul style="list-style-type: none">If getImageURL(item) returns null: check for the next extension.Else, use the URL being returned for secondary icon within visualization of node in graphical impact analysis.
Provided by	CentraSite Control
Example	<pre><extension point="com.centrasite.control.assocNavigatorNodeDecorator" id="ExternalLinkNodeDecorator" class="com.centrasite.control.ext.ExternalLinkNodeDecorator"> </extension></pre>

The picture illustrates how ExternalLinks objects are decorated with icons representing the type of the object they are referencing.



Append Root Node to Topic

Usage	Append a root node to an existing topic.
--------------	--

Attributes	■ <code>point="com.centrasite.control.topicItems"</code>
	■ <code>id</code>
	■ <code>class</code>
<hr/>	
Interface	TopicItems
	■ <code>boolean appliesTo (Topic)</code>
	■ <code>Collection getItems()</code>
<hr/>	
Abstract base class AbstractTopicItems	
<hr/>	
Processing	For each topic whose implementation class is derived from a class named BaseTopic (true for all topics contributed by CentraSite Control) it is checked whether there are any extension for the topicItems extension point. Each extension whose <code>appliesTo()</code> method returns <code>true</code> , the collection of Item objects returned by <code>getItems()</code> is appended to the set of root nodes for the corresponding topic.
<hr/>	
Provided by	CentraSite Control
<hr/>	
Example	<pre><extension point="com.centrasite.control.topicItems" id="filesystem" topic="com.centrasite.control.administration" class="com.centrasite.control.ext.junit. FileSystemTopicItems"> </extension></pre>
	Note: Here, the FileSystemTopicItems extension is an extension of the base class AbstractTopicItems whose <code>appliesTo()</code> method returns <code>true</code> if the value of the topic attribute matches the ID of the topic being passed.

Replace Standard Detail View by Another Editor

Usage	Add an editor that can be configured per object type, even per object instance.
<hr/>	
Attributes	■ <code>point="com.centrasite.control.itemEditor"</code>
	■ <code>id</code> (default: name of implementing class)
	■ <code>class</code>
<hr/>	
Interface	ItemEditor
	■ <code>public boolean appliesTo (Item item, Connector connector);</code>
	■ <code>public String getLayout();</code>
	■ <code>public String getTitle(Item item);</code>

- `public String getAdapterClass();`
(must return a class implementing the `ItemEditorAdapter` interface)

Abstract base class `AbstractItemEditor`

- | | |
|-------------------|--|
| Processing | <p>If <code>appliesTo()</code> returns true, the editor is used when opening the detail for the item being passed:</p> <ul style="list-style-type: none"> ■ The given adapter class will be instantiated and initialized. ■ The given layout (=pageURL) is opened in the CONTENT frame on the right-hand side. ■ The title returned by <code>getTitle()</code> is used as the label for the activity. |
|-------------------|--|
-

Provided by	CentraSite Control
--------------------	--------------------

Example	<pre><extension point="com.centrasite.control.itemEditor" id="DataType" class="com.softwareag.centrasite.ext.DataTypeEditor"> </extension></pre>
----------------	--

Extend Search Dialog by Additional Conditions

Usage	Extend the search dialog by additional conditions, for example, you can add specific search predicates for your own object types.
--------------	---

Attributes	<ul style="list-style-type: none"> ■ <code>point="com.centrasite.control.searchPredicate"</code> ■ <code>id</code> ■ <code>class</code>
-------------------	--

Interface	<p><code>PredicateEditor</code></p> <ul style="list-style-type: none"> ■ <code>Predicate getPredicate()</code> (Get predicate to be added by this editor) ■ <code>String getLayout()</code> (Get URL of layout to be rendered) ■ <code>String getAdapterClass()</code> (Get name of adapter class to be used for rendering, must be a subclass of <code>AbstractPredicateAdapter</code>) ■ <code>String getPredicateClass()</code> (Get name of predicate class to be used for rendering, must be a subclass of <code>AbstractPredicate</code>)
------------------	---

The interface Predicate (many implementing classes are already available in `CentraSiteUtils.jar`) with its abstract subclass `AbstractPredicate` has the following methods:

- `boolean appliesTo(String objectTypeValue, CentraSiteQueryManager qm)`
(Check whether this predicate applies to objects of given object type)
- `String getInternalType ()`
(Get unique internal string representation of type of predicate; not to be localized. You may use a namespace-like notation for your own.)
- `String getDisplayType ()`
(Get human readable localized representation of type of predicate; CentraSite Control will display it on the left hand side in the **Add Condition** dialog)
- `void validate() throws InvalidPredicateException`
(Validate parameters set for this predicate. The `InvalidPredicateException` should contain a localized message text)
- `String getDisplayString () throws Exception`
(Get human readable localized string representation of predicate including values predicate; CentraSite Control displays it in the condition table in the header section of the **Search Registry** dialog)
- `boolean requiresEnterpriseLicense()`
(Check whether this predicate requires an Enterprise license)
- `void addTo (BusinessQuery bq) throws JAXRException`
(Add contribution of predicate to given `BusinessQuery`. This is the worker method applying the predicate to the search result.)

`AbstractPredicate` also provides implementations for the following methods:

- Used for I18N support
- `Locale getLocale()`
- `Void setLocale(Locale)`
- used for persisting predicates as part of queries.
- `String toXML ();`
- `void setFromDom(Element predicateElement, Connection connection)`

- used to initialize the search dialog with read only predicates or conditions, which can not be modified or removed:
- `void setReadOnly(boolean readOnly);`
- `boolean isReadOnly();`

Abstract base class AbstractPredicateEditor

- | | |
|-------------------|---|
| Processing | <ul style="list-style-type: none"> ■ When you click the appropriate button, this invokes the user-defined Adapter (layout) screen for entering custom search related settings. ■ Create an instance of the class ■ Execute |
|-------------------|---|
-

Provided by	CentraSite Control
--------------------	--------------------

Example	<pre><extension point="com.centrasite.control.searchPredicate" id="ObjectTypePredicateEditor" class="com.centrasite.control...ObjectTypePredicateEditor"> </extension></pre>
----------------	---

Download Documents

There is a menu entry in each asset's context menu that allows you to create a zipped archive of the asset and optionally any attached documents, and to download the zipped archive to the file system. You can customize the way in which the download feature behaves:

- You can make the download entry in the context menu visible or invisible for users with the Guest role.
- You can change the text string displayed in the context menu.
- You can change the format of the zipped archive.

If a user with the Guest role can access an asset and view its context menu, the context menu entry **Download Document** is visible by default. You can specify whether this entry is visible or invisible for such users as follows:

➤ To make the download menu entry visible or invisible for guest users

1. Locate the configuration file `plugin.xml` in the `<RuntimeDir>\workspace\webapps\PluggableUI\CentraSiteControl` directory.
2. Open the file and locate the following entry:

```
<extension point=com.softwareag.cis.plugin.parameter
  id=guestCanDownloadDocuments value=true/>
```

3. To make the context menu entry invisible for guest users, change `true` to `false` and restart Software AG Runtime. Similarly, if the context menu entry is already invisible and you want to make it visible for guest users, set the value to `true` and restart Software AG Runtime.

Changing the text string displayed in the Context menu

The text string displayed in the context menu is by default `Download Document`. You can change this by extending the CentraSite Control functionality through the extension point `downloadDocumentCommand`. This extension point has the following definition:

Usage	Change the text string displayed in the context menu for downloading an asset.
Attributes	<ul style="list-style-type: none">■ <code>com.centrasite.control.downloadDocumentCommand</code>■ <code>id</code> (default: name of implementing class)■ <code>class</code>
Interface	See the sample code.
Abstract base class	<code>AbstractExtensionCommand</code>
Provided by	CentraSite Control
Example	See the sample code.

To change the text displayed for the context menu, the implementation of the extension point must define a method `getName()` of type `String`. The return value of this method is the text that will be displayed in the context menu.

You can find sample code for defining the extension point in the file `DownloadDocumentCustomCommand.java` that is provided in the `demo` folder under the CentraSite installation folder.

Changing the format of the zipped archive

By default, the zipped archive contains the folder structure of the asset and its attached documents. You can change this by extending the CentraSite Control functionality through the extension point `downloadDocumentCommand`.

To change the format of the zipped archive, your implementation of the extension point must define a method that extends the base class `DownloadOperation`.

You can find sample code for defining the extension point in the files `DownloadDocumentCustomCommand.java` and `DownloadCustomOperation.java` that are provided in the `demo` folder under the CentraSite installation folder.

Attach Documents

Some assets include file-related attributes that allow you to attach supporting document(s) such as programming guides, sample code, and script files with the asset. While trying to attach a supporting document with an asset, CentraSite Control displays the available documents

underneath their respective organization directory by default. You can change this (that is, simply display the documents by the side of its organization directory) by extending the CentraSite Control functionality through the extension point `attachDocumentCommand`.

➤ To customize the document layout

1. Locate the configuration file `plugin.xml` in the `<RuntimeDir>\workspace\webapps\PluggableUI\CentraSiteControl` directory.
2. Open the file and locate the following entry:

```
<extension point="com.softwareag.cis.plugin.parameter"
  id="isCustomAttachDocument" value="false" />

<extension point="com.centrasite.control.attachDocumentCommand"
  id="AttachDocumentCustomCommand"
  class="com.centrasite.control.extpt.AttachDocumentCustomCommand"
/>
```

Where `com.centrasite.control.extpt.AttachDocumentCustomCommand` is the name of the abstract base class that implements the interface.

3. To define your custom document layout, change `false` to `true` and restart Software AG Runtime. Similarly, if the document layout is already customized and you want to revert to the standard layout, set the value to `false` and restart Software AG Runtime.

Usage	Use this to define a custom layout of the documents while attaching to an asset through the Attach Document dialog.
Attributes	<ul style="list-style-type: none"> ■ <code>com.centrasite.control.attachDocumentCommand</code> ■ <code>id</code> (default: name of implementing class) ■ <code>class</code>
Interface	See the sample code.
Abstract base class	<code>AbstractExtensionCommand</code>
Processing	<p>When you click the appropriate button, this invokes the user-defined Adapter (layout) screen displaying all documents that are available for attaching to an asset.</p> <ul style="list-style-type: none"> ■ Create an instance of the class ■ Execute
Provided by	CentraSite Control
Example	<pre><extension point="com.centrasite.control.attachDocumentCommand" id="AttachDocumentCustomCommand" class="com.centrasite.control. extpt.AttachDocumentCustomCommand" /></pre>

You can find sample code for defining the extension point in the files `AttachDocumentCustomCommand.java`, `AttachFile.xml`, and `AttachFileAdapter.java` that are provided in the demos folder under the CentraSite installation folder.

Activating the IDE

The CentraSite distribution kit contains an IDE (integrated development environment) that you can use to create and design a layout page. The IDE is a web application whose clients run on a web browser. The URL (assuming installation defaults) to start the IDE on a machine where CentraSite is installed is:

`http://localhost:53307/PluggableUI/HTMLBasedGUI/workplace/ide.html`

The IDE is deactivated by default. In order to activate the IDE, set the attribute `plugindevelopment` in the file `cisconfig.xml` to `true`. This file is located in the CentraSite Control web application (in the Application Server or Software AG Runtime location) in the folder `cis/cisconfig`.

The following example illustrates the required configuration setting:

```
<cisconfig plugindevelopment="true" ...>
...
</cisconfig>
```

Security Considerations

When activated, the IDE and included development tools do not require further authentication. The following example illustrates the `security-constraint` and `login-config` elements to protect the IDE and development tools with the HTML basic authentication method.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Plugin Development</web-resource-name>
    <url-pattern>/HTMLBasedGUI/workplace/*</url-pattern>
    <url-pattern>/servlet/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>developer</role-name>
  </auth-constraint>
</security-constraint>
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Plugin Development</realm-name>
</login-config>
```

In order to protect passwords transmitted in clear text between a browser and development tools running on the application server, it is recommended to protect the communication through the use of SSL. For more information about configuring secure communication between CentraSite components, see *CentraSite Administrator's Guide*.

Setting Preferred Plug-in and Order of Plug-ins

You can adapt the URL used to invoke the pluggable user interface with a preferred plug-in by appending a query parameter such as

```
PLUGIN=com.centrasite.control
```

So the modified URL will be:

```
http://localhost:53307/PluggableUI/servlet/StartCISPage?PAGEURL=/PluggableUI/
Login.html&PLUGIN=com.centrasite.control&LOCALE=en
```

The value of the `PLUGIN` parameter must match the value of the `id` parameter of a `<plugin>` root element in one of the plug-ins. This sets the preferred plug-in.

This implies that for all extensions for a specific extension point, the extensions belonging to the referenced plug-in is first in order (normally the order is determined by the `order` attribute in the plug-in configuration file).

For any extension point, the order of the associated extensions is determined by the following properties:

- The processing order of the plug-ins is controlled by the value of the `order` attribute of `<plugin>` in the `plugin.xml` file. Plug-ins with a smaller value of the `order` attribute are processed first. The preferred plug-in is always processed first.
- The order of extensions, as configured in `plugin.xml`, for the associated extension point.

Depending on the extension point, the order of the extensions has a specific impact, for example:

- The login screen displayed when the user interface is started in the browser.
- The initial perspective shown after login.

Installing and Uninstalling Plug-ins

Directory Structure

The plug-in environment is contained in a directory structure under the installation directory `<RuntimeWebAppsDir>` of the Software AG Runtime.

Under `<RuntimeWebAppsDir>\PluggableUI` we have the following structure:

```
WEB-INF/
  classes/
    log4j.xml
  lib/           //JARs

cis/

HTMLBasedGUI/

PluggableUI
  plugin.xml
  *.html

  accesspath/
  xml/         // layout definitions
  images/

CentraSiteControl
  plugin.xml
```

```
*.html
*_SWT.xml

accesspath/
xml/        // layout definitions
images/     // icons
lib/        // JARs
classes/    // class files

MyPlugIn
plugin.xml
*.html
*_SWT.xml

accesspath/
xml/        // layout definitions
images/     // icons
lib/        // JARs
classes/    // class files
```

The structure includes a sample user-written plug-in `MyPlugIn` for illustration purposes.

➤ Installing a Plug-in

A plug-in should be provided as a ZIP archive with the directory structure.

The following actions need to be performed when installing a plug-in manually:

- Check for availability of other plug-ins.
- Copy files (except the `plugin.xml` configuration file) into the directory structure.
- Compile layout definitions.

Note:

Using the plug-in may require a restart of the Software AG Runtime.

➤ Uninstalling a Plug-in

The following actions need to be performed to uninstall a plug-in manually:

- Before you uninstall a plug-in, ensure that it is not required by another plug-in.
- Remove the plug-in configuration file `plugin.xml`.
- Remove the plug-in directory, for example `MyPlugIn`.

Note:

It might not be possible to remove files if they are in use, for example, while the application server is running.

Plug-In Management Perspective

A separate Plug-In Management perspective offers the following functions:

Function	Description	Invoke via...
Install Plug-In	Import a ZIP-file containing all required files for a plug-in	Button in toolbar
Table of Plug-ins	Similar to the About dialog	Select a node in the Plug-Ins topic
Uninstall Plug-In	Check which other plug-ins rely on the plug-in to be uninstalled. If there are no dependencies, the plug-in is uninstalled.	Select the plug-in in the table and select the command from the context menu
Compile Layouts	Required when the underlying Application Designer runtime is upgraded	Select plug-in in table and select command from context menu
Start the Application Designer layout editor		Button in toolbar

The Plug-In Management perspective is not visible by default. It is only visible if you set the preferred plug-in using `PLUGIN=com.softwareag.cis.plugin` in the URL that is used to start the GUI.

Example:

```
http://localhost:53307/PluggableUI/
servlet/StartCISPage?PAGEURL=/PluggableUI/Login.html&PLUGIN=com.softwareag.cis.plugin&LOCALE=en
```

Special and Advanced Topics

Icons

There are various optional references to icons which may be contributed by a plug-in. Most icons should be transparent GIFs unless stated otherwise in the table below. Here is a set of potential locations for contributing icons:

Context	Recommended Size in Pixels	Remarks
Plug-in icon appearing in the common About dialog	16x16	Transparent GIF
Bitmap for plug-in specific 2nd-level About dialog	None	May be also JPG or PNG file
Perspective icon in Select Perspective dialog	16x16	Transparent GIF

Context	Recommended Size in Pixels	Remarks
Header icon contributed by perspective	Height: 35. Width: depends on space required for toolbar and view listbox.	May be also JPG or PNG file
Icon representing an item in tree or table	16x16	Transparent GIF
Icon for command for an item (context menu or toolbar in detail view)	16x16	Transparent GIF

Class Loading

The Pluggable UI depends on dedicated class loaders. The code for a normal web application is only loaded through the basic `WebappClassLoader` provided by the servlet container (for example, Tomcat), this class loader is only used for loading the classes resembling the PluggableUI base with the underlying Application Designer. The respective classes are loaded from the following directories below `<RuntimeWebAppsDir>\PluggableUI`:

```
WEB-INF/classes WEB-INF/lib/*.jar
```

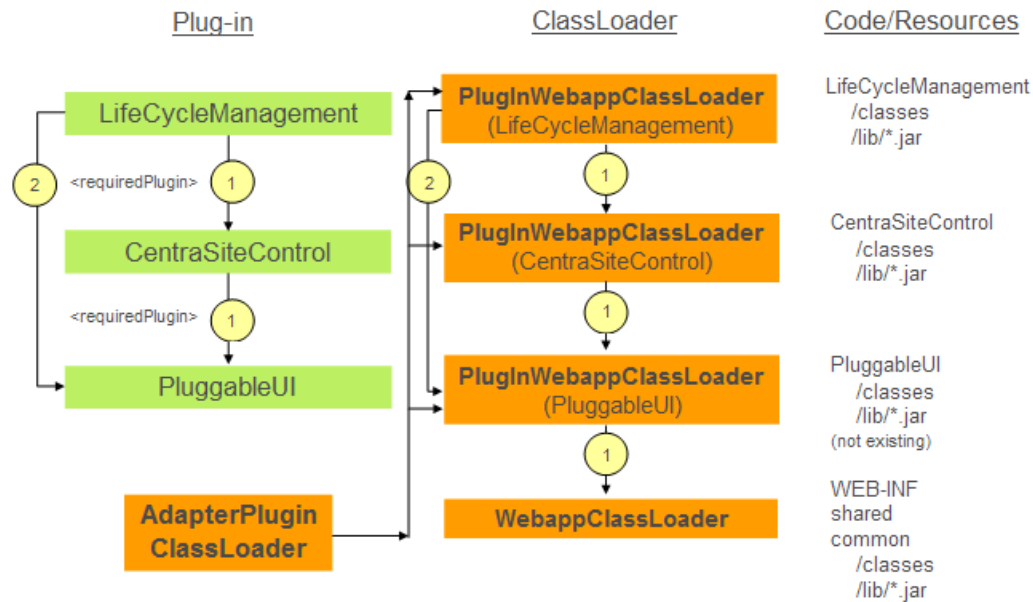
In addition, locations holding common or shared class or jar files are searched by the `WebappClassLoader` when attempting to resolve references to required classes.

Any code contributed by a plug-in is loaded by a corresponding instance of the `PlugInWebappClassLoader` from the following directories below `<RuntimeWebAppsDir>/PluggableUI`:

```
plugInDir/classes plugInDir/lib/*.jar
```

If resolution fails, the `PlugInWebappClassLoader` for the current plug-in delegates class loading to the `PlugInWebappClassLoaders` for other plug-ins in the order listed as `<requiredPlugin>` in the `plugin.xml` of the current plug-in recursively. Finally, if resolution through required plug-in fails, the `PlugInWebappClassLoader` delegates class loading to the `WebappClassLoader`.

The following picture illustrates the scenario of the `LifeCycleManagement` plug-in (representing any other third party plug-in) that depends on the `CentraSiteControl` plug-in and the PluggableUI base infrastructure.



The AdapterPluginClassLoader is used by Application Designer when resolving references to adapter classes found in layout definitions. The AdapterPluginClassLoader never loads classes itself. Instead, it asks all known PluginWebappClassLoaders in an unspecified order to load a required class.

CAUTION:

You must avoid having the same adapter classes in more than one plug-in, else it results in various classloading related issues (ClassCastException, ClassNotFoundException, ...). Under normal conditions, fulfilling this restriction should not cause any problems.

In general, you should avoid having multiple locations contributing the same classes within the graph of locations spanned by the required plug-ins.

Multithreading and Synchronization

Normally, when executing the HTTP requests on behalf of a single Application Designer session, there is no parallel execution in multiple threads (unless the code contributed by a plug-in starts a thread on its own). Hence, access to objects or properties having a scope restricted to the session context does not require any synchronization.

However, when using global or static variables, this is no longer true. Multiple active user sessions may be processed in parallel.

Avoid usage of global variables, which might lead to the following issues:

- Synchronization is required otherwise non-reproducible race conditions occur.
- Memory leakages if global collections that grow for each session are used.

- Global references to JAXR-based RegistryObjects. A RegistryObject contains a reference to the JAXR-based Connection (including the underlying credentials) that had been used to load it. When resolving a secondary reference either of the following may happen:
 - If the connection is still open, credentials of another user is used, thus causing a security hole.
 - If the connection is no longer open, a corresponding exception occurs (trying to use a closed connection).

Nested Layouts

All adapter classes of a plug-in should not be just subclasses of `com.softwareag.cis.server.Adapter`. Instead, they should be derived from one of the following classes:

- `com.softwareag.cis.plugin.adapter.util.CommonAdapter` - for a plug-in that does not depend on CentraSite Control
- `com.centrasite.control.adapters.BaseAdapter` - for a plug-in that depends on CentraSite Control

`BaseAdapter` is a subclass of `CommonAdapter` and thus inherits certain properties. Among those is the implicit registration of all adapters as known adapters in the current session context. However, under certain circumstances it may happen that adapters for nested pages displayed using a `SUBCISPAGE` or `ROWTABSUBPAGES` control are not automatically deregistered when closing, for example, an activity displayed in the `CONTENT` frame on the right hand side of the workplace. This may lead to subsequent `NullPointerExceptions`.

Normally, deregistration is accomplished within the `destroy()` method in `CommonAdapter`. Hence, be careful when overriding this method in a subclass to call `super.destroy()`. In addition, you should override the `endProcess()` method in the adapter for the container layout, which should perform at least the following actions:

- call `super.endProcess()`
- call `CommonAdapter.removeKnownAdapter (subPageAdapter)` for each `subPageAdapter`

Creating a Sample Customized Plug-in for Content Pages

This section describes how to create customized plug-ins for the CentraSite Control Content pages. The descriptions in this topic are based on a sample plug-in named `DemoPlugIn01`. It describes how to use Eclipse and standard CentraSite features in order to add the plugin to CentraSite.

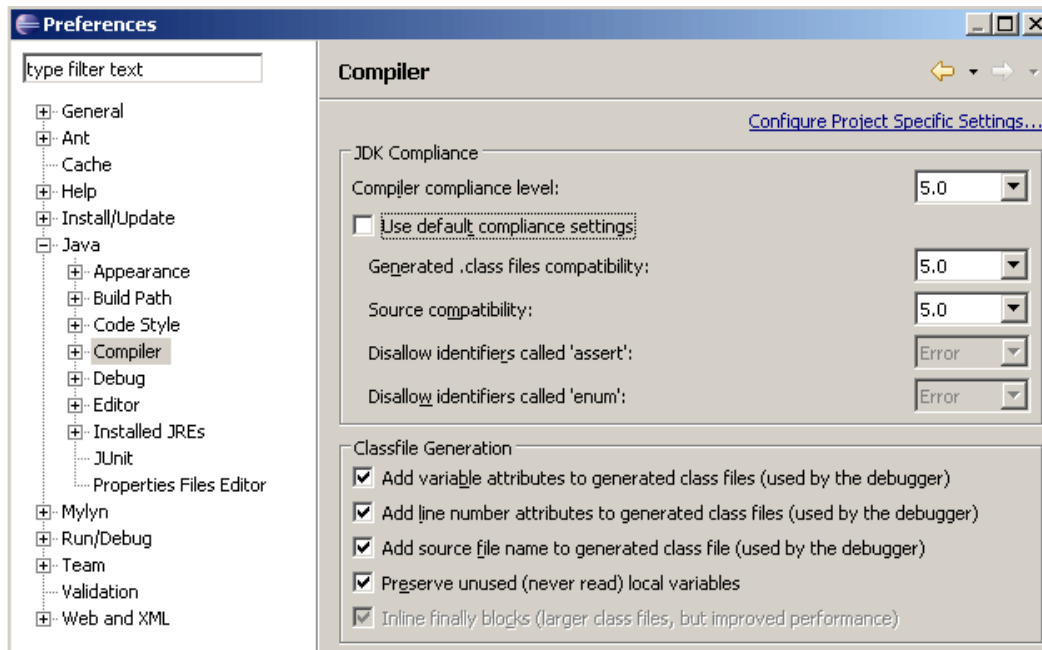
Eclipse Prerequisites

Before you start, ensure that you have a recent Eclipse version installed on your machine.

In Eclipse, select **Window > Preferences > Java > Compiler** in order to configure usage of / compliance with the Java version currently supported by CentraSite.

You can check the system requirements at <http://documentation.softwareag.com>.

Example:



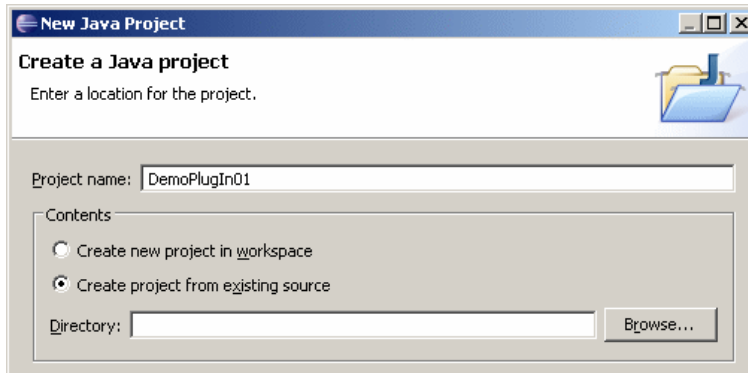
Click **Apply** to activate the settings. Eclipse asks you to confirm the change, indicating that an internal rebuild is required. Click **Yes**. The rebuild takes a few seconds.

Setting up the Plug-in Project

As part of setting up a plug-in project you have to create a new Java project, adapt it to suit your environment, and add icons required for the plug-in.

➤ To setup the plug-in project

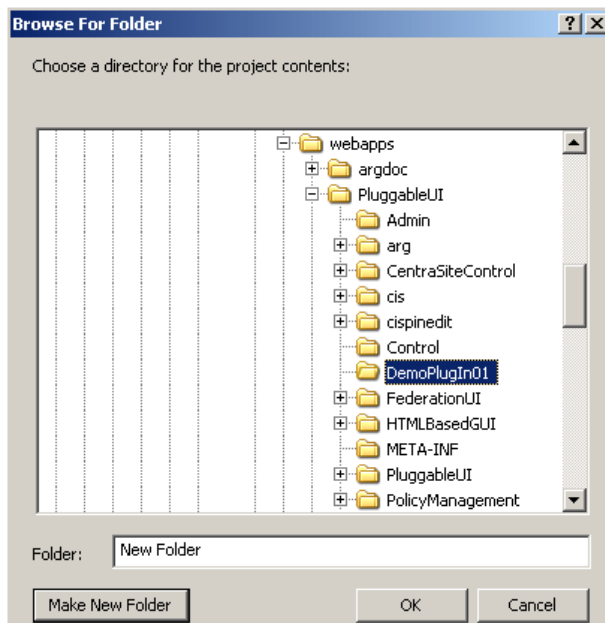
1. Create a new Java Project in Eclipse.
 - a. Click **File > New > Project > Java Project**.
 - a. Specify DemoPlugIn01 as the project name and select **Create project from existing source**.



- b. Click **Browse**.
- c. In the **Browse For Folder** dialog, navigate to and click on the `PluggableUI` web application folder of the Software AG Runtime application. In the remainder of this document, this folder is indicated by `<PluggableUIFolder>`.
- d. Click **Make New Folder**.

This creates an entry `New Folder` under `PluggableUI`.

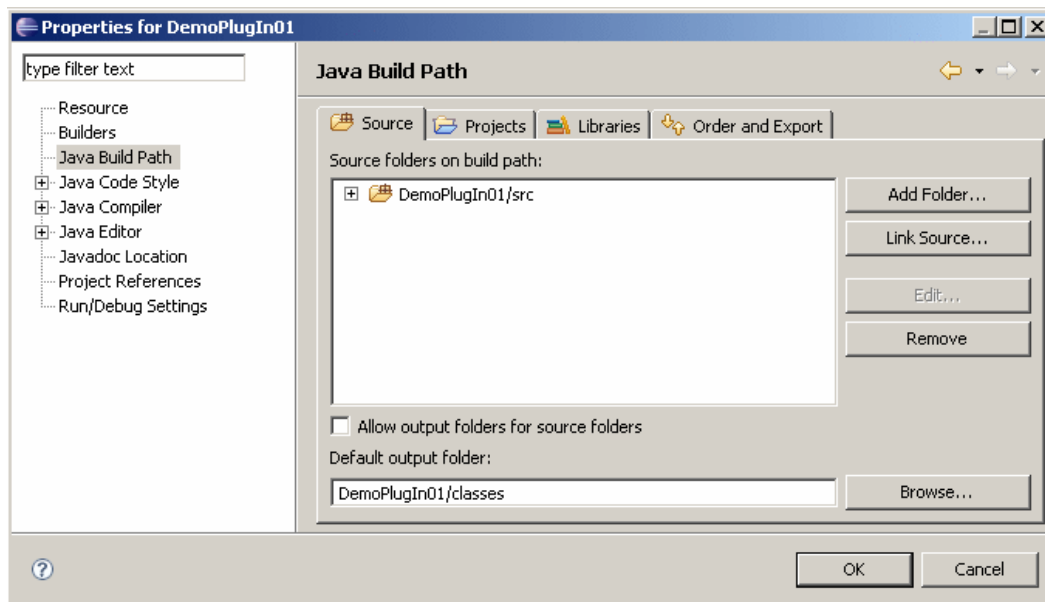
- e. Select `New Folder`, and from its context menu select **Rename** and type the name `DemoPlugIn01`.



- f. Click **OK**.
- g. In the **New Java Project** dialog, click **Finish**.

A new Java project called DemoPlugIn01 is created. This project is now visible in the **Package Explorer** view in Eclipse.

2. Adapt the project to suit the environment.
 - a. Create the four subfolders accesspath, classes, images, and xml by selecting **New > Folder** from the context menu of DemoPlugIn01 in the package explorer and typing the name in the **New Folder** dialog.
 - b. Create a source folder called src by selecting **New > Source Folder** from the context menu of DemoPlugIn01.
 - c. In the context menu of DemoPlugIn01, select **Properties**.
 - d. Select **Java Build Path** in the left navigation tree.
 - e. Select the **Source** tab and type the value DemoPlugIn01/classes in the field **Default output folder**.



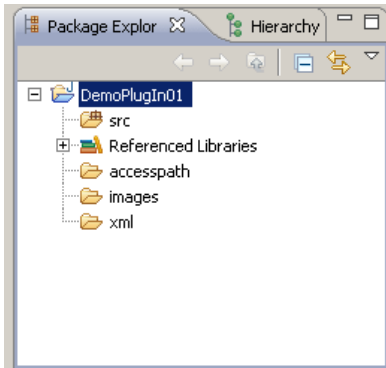
- f. Click the **Libraries** tab.

An entry for the JRE library should be visible. If you do not see this entry, click **Add Library**, select the JRE system library from the displayed list, and click **Finish**.

- g. Click **Add External JARs**.
 - h. In the **JAR Selection** dialog, navigate to `<PluggableUIFolder>/WEB-INF/lib` and open the folder.

- i. Select all files and click **Open**.
- j. Click **Add External JARs** again.
- k. In the **JAR Selection** dialog, navigate to `<PluggableUIFolder>/CentraSiteControl/lib` and open the folder.
- l. Select all files and click **Open**.
- m. Click **OK** in the **Properties for DemoPlugIn01** dialog.

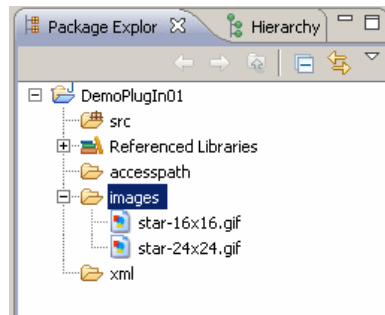
Your project should now look like this:



Note:

The `classes` subfolder of the project `DemoPlugIn01` does not appear. This is normal because the Java Development Tools (JDT) of Eclipse suppress output folders from displaying by default (but they still exist on your hard disk). Also, the old output folder `bin` that has been created by the JDT when creating the Java project is not of any use, so you can delete it.

3. Add icons required for the plug-in. Here we are copying and renaming some already existing icons from the CentraSite Control plug-in and using them.
 - a. Using Windows Explorer, navigate to `<PluggableUIFolder>/CentraSiteControl/images`.
 - b. Copy the files `myFavorites.gif` and `myFavorites24x24.gif` to the `images` subfolder of the Java project `DemoPlugIn01`.
 - c. In `DemoPlugIn01/images`, rename the file `myFavorites.gif` to `star-16x16.gif` and rename `myFavorites24x24.gif` to `star-24x24.gif`. Use the command **File > Rename** in the Eclipse menu to do this.
 - d. In Eclipse, refresh the display of the package explorer. The names of the two images should now be visible.



Plugging into CentraSite Control

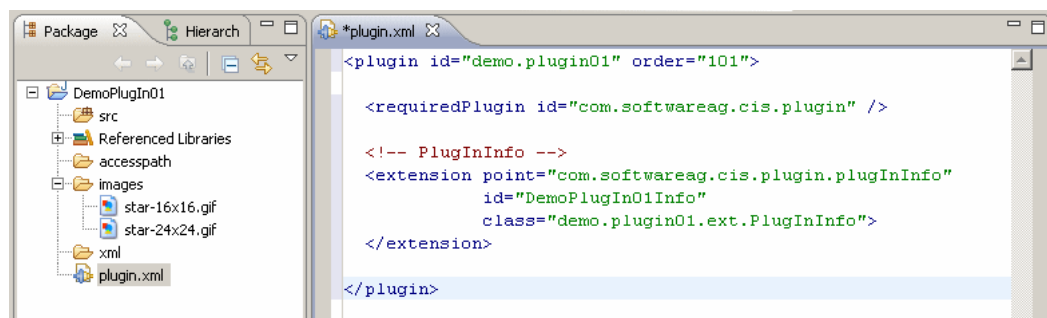
You have to plug-in into CentraSite Control to let it know the presence of the plug-in configuration file. The plug-in configuration file contains the information about where a plug-in plugs into CentraSite Control. The CentraSite Control software provides extension points. These are positions in the program logic of the CentraSite Control program where functionality can be added by a plug-in. Every time the program flow comes to such an extension point, a search for plug-ins that extend CentraSite Control at this point takes place and the code provided by the plug-ins is invoked.

1. Convert the Java project to a CentraSite Control plug-in folder by providing a plug-in configuration file
 - a. In the context menu of DemoPlugIn01 in the package explorer, select **New > File**.
 - b. Type plugin.xml as the file name and click **Finish**.
 - c. Type the following XML code:

```
<plugin id="demo.plugin01" order="101">

  <requiredPlugin id="com.softwareag.cis.plugin" />

  <!-- PlugInInfo -->
  <extension point="com.softwareag.cis.plugin.pluginInfo"
    id="DemoPlugIn01Info"
    class="demo.plugin01.ext.PlugInInfo">
  </extension>
</plugin>
```



- d. Save the file using **Ctrl+S**.

A plug-in must have an identifier (here `demo.plugin01`) which has to be unique among all plug-ins. It is recommended to use naming conventions similar to Java package names. The order number of a plug-in (here `101`) gives CentraSite Control a priority for the sequence in which the plug-ins have to be loaded at startup. The higher the number, the later a plug-in is loaded. We have to declare the plug-in as being dependent on the `com.softwareag.cis.plugin` because we use an extension point provided by this plug-in. This dependency is indicated through the `requiredPlugin` XML element. The extension XML element in the file `DemoPlugIn01/plugin.xml` denotes that our plug-in extends the user interface at a point where information about a plug-in can be contributed. The string that looks like a Java package name is the name of the extension point (`com.softwareag.cis.plugin.pluginInfo`). The extension identifier (here `DemoPlugIn01Info`) must be unique among all extension identifiers of a plug-in. The `class` attribute specifies the fully qualified name of the class that implements the extension (here `demo.plugin01.ext.PluginInfo`). The top level package name for all of our Java code is `demo.plugin01`. Select `ext` as the subpackage name for the implementing class to denote that code that extends CentraSite Control resides here.

2. Implement the extension, that is, provide a Java class called `demo.plugin01.ext.PluginInfo`, which implements a specific interface required by the extension point.
 - a. In the context menu of `DemoPlugIn01/src` in the package explorer, select **New > Class**.
 - b. Specify `demo.plugin01.ext` for the package name, `PluginInfo` for the class name and `com.softwareag.cis.plugin.extpt.util.AbstractPluginInfo` for the superclass (you can use the **Browse** button to avoid typing).
 - c. Make sure that the check box labeled with **Inherited abstract methods** is selected and click **Finish**.

The image shows the 'New Class' wizard in Eclipse. The 'Source folder' is 'DemoPlugIn01/src', 'Package' is 'demo.plugin01.ext', and 'Enclosing type' is empty. The 'Name' is 'PlugInInfo'. Under 'Modifiers', 'public' is selected. Under 'Superclass', 'com.softwareag.cis.plugin.extpt.util.AbstractPlugInInfo' is entered. The 'Which method stubs would you like to create?' section has 'Inherited abstract methods' checked. At the bottom are 'Finish' and 'Cancel' buttons.

Eclipse now opens the file `PlugInInfo.java` in the Java editor.

- d. Modify `PlugInInfo.java` in the Java editor as follows:

```
package demo.plugin01.ext;

import com.softwareag.cis.plugin.extpt.util.AbstractPlugInInfo;

public class PlugInInfo extends AbstractPlugInInfo {

    public String getImageURL() {
        return "../DemoPlugIn01/images/star-16x16.gif";
    }

    public String getLayout() {
        return null;
    }

    public String getTitle() {
        return "DemoPlugIn01";
    }
    public String getVendor() {
        return "Software AG";
    }

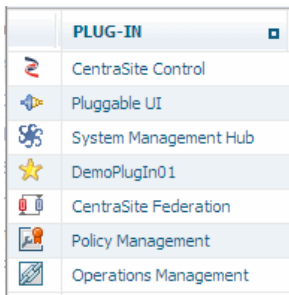
    public String getVersion() {
        return "0.0.0.1";
    }
}
```

- e. Save the modified file and ensure that no compilation errors occur.

When you save the file, the Java file is automatically compiled into the folder `classes` of the project `DemoPlugIn01`. (Remember, the `classes` subfolder of the project is suppressed from displaying). The resulting class file is now accessible for the pluggable user interface of CentraSite Control.

3. Check if CentraSite Control is aware of the plug-in.
 - a. Restart the Windows service Software AG Runtime.
 - b. Start the CentraSite Control application from the Windows **Start > All Programs > Software AG** menu, and log in using your credentials for ID and password.
 - c. Click the **About** button at the top of the page and click **Plug-Ins**.

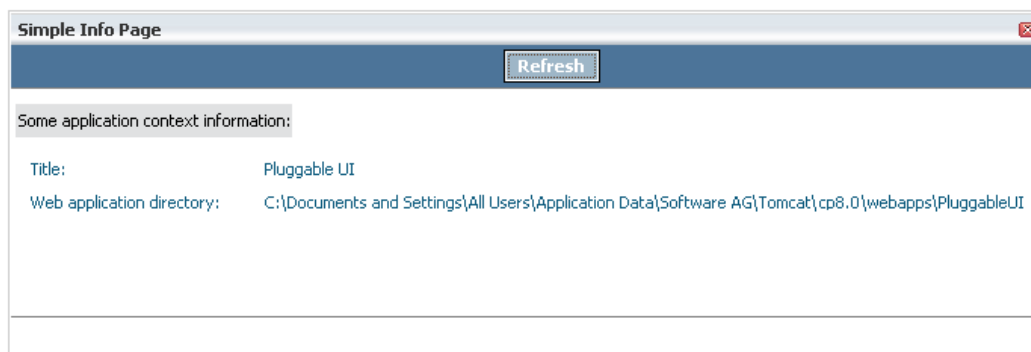
If everything works fine, a dialog box whose contents look quite similar to the following figure appears. In particular, the line that represents the sample plug-in `DemoPlugIn01` should be visible:



Including Your Own Layouts to the Screen

You can extend CentraSite Control by embedding customized layout pages. This section describes how to create a layout page using the Application Designer IDE and include them onto the screen.

The figure depicts the final result of this procedure:

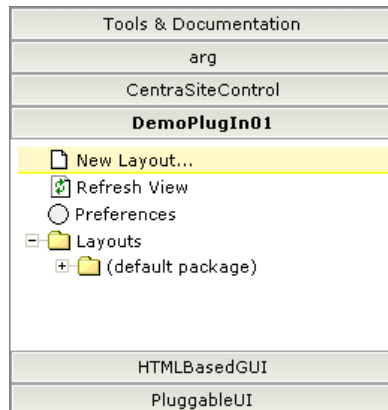


The CentraSite distribution kit contains an IDE (integrated development environment) that you can use for creating a layout. The IDE is a web application whose clients run on a web browser. The URL (assuming installation defaults) to start the IDE on a machine where CentraSite is installed

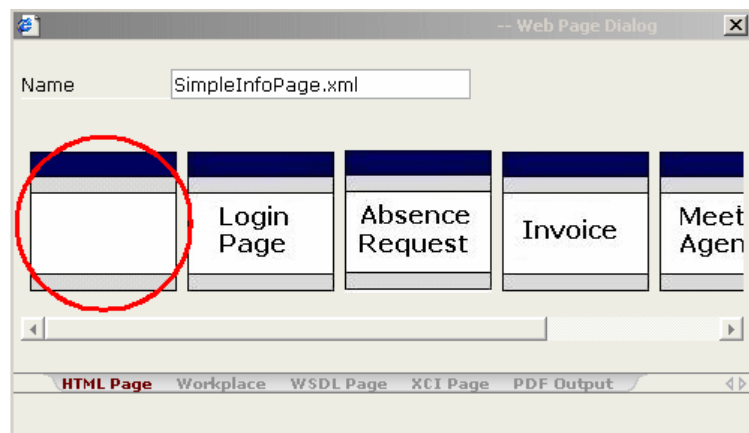
is `http://localhost:53307/PluggableUI/HTMLBasedGUI/workplace/ide.html`. There is no shortcut created by the installation to start IDE. Therefore you have to create one manually.

The IDE delivered in the distribution kit needs to be activated before you can use it. For information on how to do this, see .

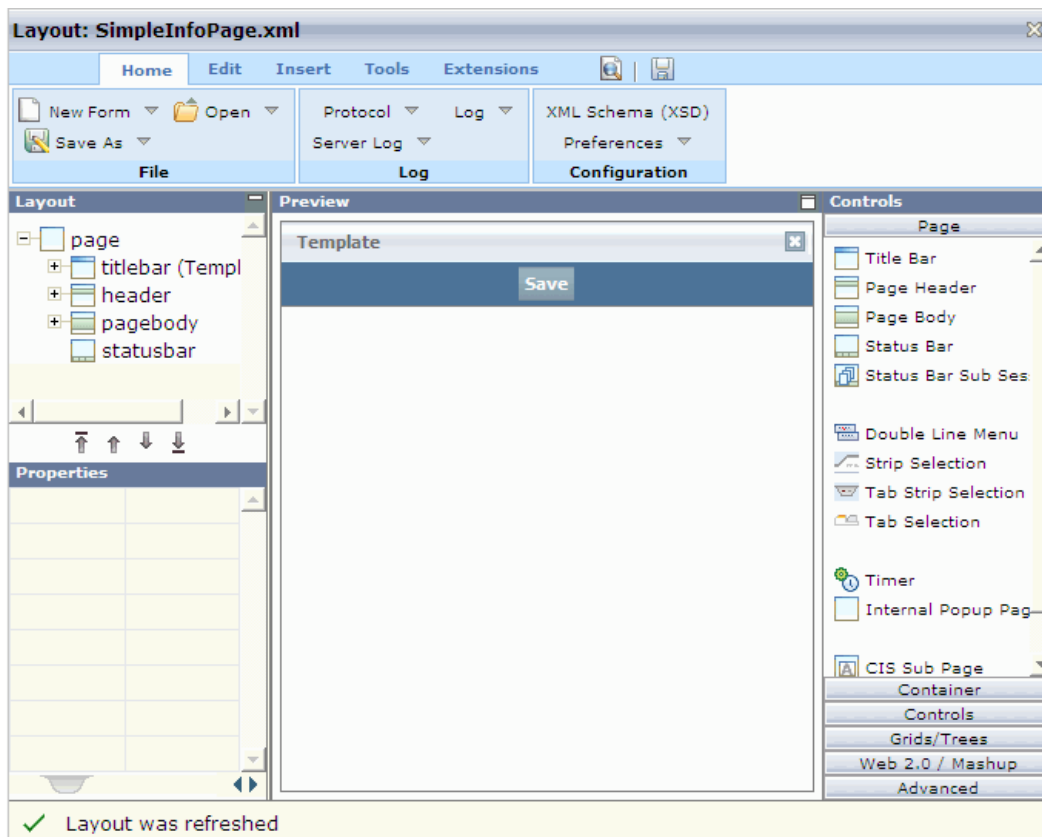
1. Create the layout page.
 - a. Start the IDE and leave the Eclipse instance running.
 - b. Click the **DemoPlugIn01** button in the button list displayed on the left.



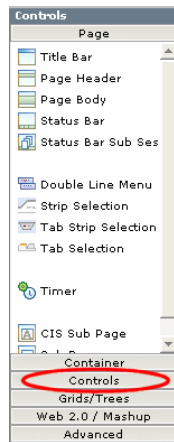
- c. Click **New Layout** on the left side directly below the **DemoPlugIn01** button.
 - d. Type `SimpleInfoPage.xml` in the input field labeled **Name** in the resulting dialog.
 - e. Click the leftmost image below the input field, as shown, to create an HTML page.



The IDE presents a standard HTML page in the preview area of the layout painter. This displays how the newly created layout looks. To view the preview, select the **Preview** icon from the toolbar of the layout painter (located beside the diskette symbol). The current look of layout `SimpleInfoPage.xml` is displayed in the preview area.

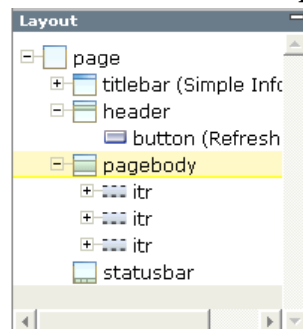


2. Design the layout as required.
 - a. Click on the title bar of the layout.
 - b. In the **Properties** view for the title bar, located at the lower left corner of the layout painter, change the name property from Template to Simple Info Page.
 - c. Click **Save**.
 - d. Change the name property from Save to Refresh and set the method property to onRefresh.
 - e. Click the diskette symbol in the tool bar of the layout painter to save the layout. The preview of the layout changes and now reflects the properties that were modified.
 - f. Click **Controls** in the **Controls** view (located right to the preview area).



- g. Add three **Independent Row** controls to the body of the page:
 - a. Click **Independent Row** and hold the left mouse button down.
 - b. Drag the **Independent Row** icon to the page body of the layout (the white area below the button that is now labeled **Refresh**) and release the left mouse button.
 - c. Click **Add as Subnode** from the popup menu that appears.
 - d. Perform the same action to add a second and a third **Independent Row** control to the page body and click **Add as last Subnode** from the popup menu that appears after releasing the mouse button.

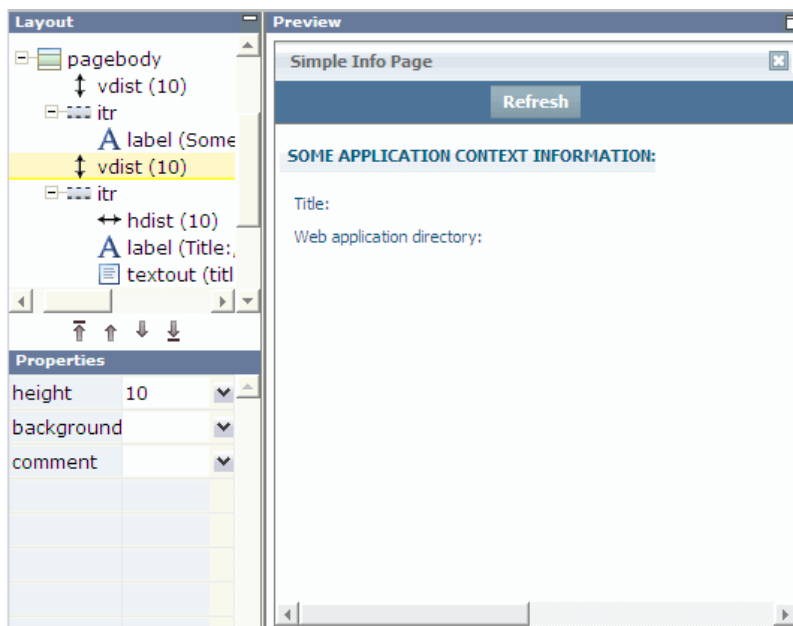
The pagebody node of the **Layout** view (located above the **Properties** view) now contains three `itr` subnodes, representing the three **Independent Row** controls.



- h. From the **Controls** view, drag and drop a **Label** control onto the first `itr` sub node of the pagebody node in the **Layout** view.
- i. In the **Properties** view set the name property to Some application context information.
- j. Set the `asheadline` property of the label to `true`.

To access this property, you have to select the **Appearance** tab at the bottom of the **Property** view. You can select the value `true` using the combo box to the right of the property name.

- k. Drag and drop a **Horizontal Distance** control onto the second `itr` sub node of the `pagebody` node in the **Layout** view.
- l. In the **Properties** view, set the width property for the **Horizontal Distance** control to 10 by just typing it in.
- m. Drag and drop a **Label** control onto the second `itr` sub node of the `pagebody` node in the **Layout** view.
- n. From the popup menu that appears after you release the mouse button, click **Add as last Subnode**.
- o. Set the name property of the newly added label to `Title:` and the width property to 200.
- p. Add a **Dynamic Text** control as the last sub node to the second `itr` sub node of `pagebody`. Set the `valueprop` property to `title` and the width property to 500.
- q. Execute the last five steps again for the third `itr` sub node of the `pagebody` node in the **Layout** view. Set the name property of the **Label** control to `Web application directory:` and the `valueprop` property of the **Dynamic Text** control to `webAppDir`. All width properties remain the same as for the children of the second `itr` sub node of `pagebody`.
- r. Surround the first `itr` sub node with two vertical distances by dragging and dropping two **Vertical Distance** controls onto the first `itr` sub node of `pagebody` (one as a preceding node and one as a subsequent node of the `itr`). Set the height property for each **Vertical Distance** control to 10.
- s. Click on the diskette symbol in the tool bar of the layout painter to save the layout.



The layout looks as required.

3. Create an adapter for SimpleInfoPage:

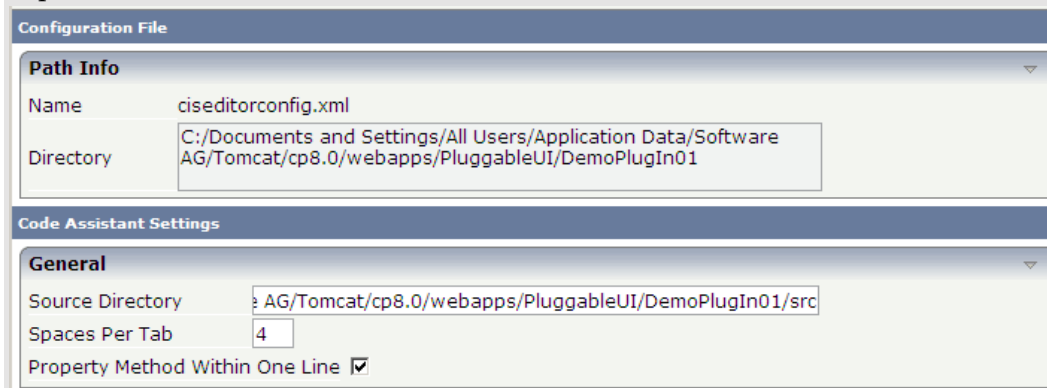
Each layout needs to have some code behind it (called the page adapter) which is not provided yet. In the page adapter you can specify how to react on events that occur due to user interactions (the push of a button for example) or fill the controls with application specific values, and so on.

The code behind the layout at this stage is provided by a dummy adapter that comes with the IDE. But you have to provide your own so that the adapter knows what to do when a user presses the **Refresh** button, for example.

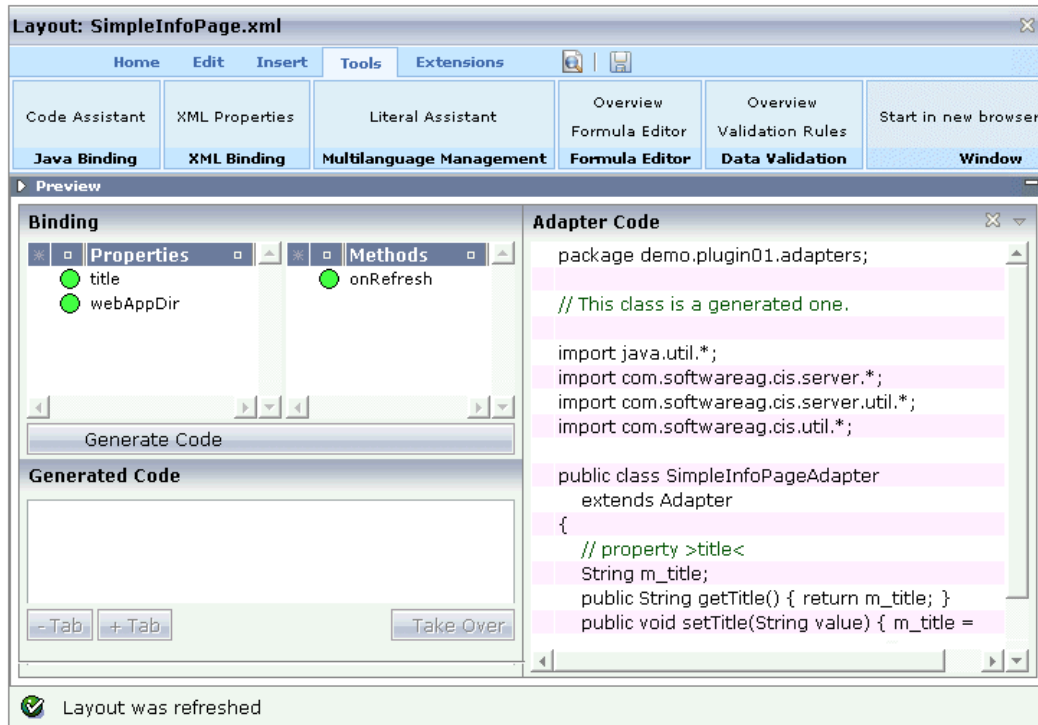
- a. Navigate to the **Home** tab of the layout painter.
- b. Select **Preferences** and type the absolute path for the Java source directory of the CentraSite Control plug-in DemoPlugIn01.

Tip:

Instead of typing the complete path, you can copy and paste the content of the field labeled **Directory** in the input field for the source directory and then append `/src` to the copied content as shown.



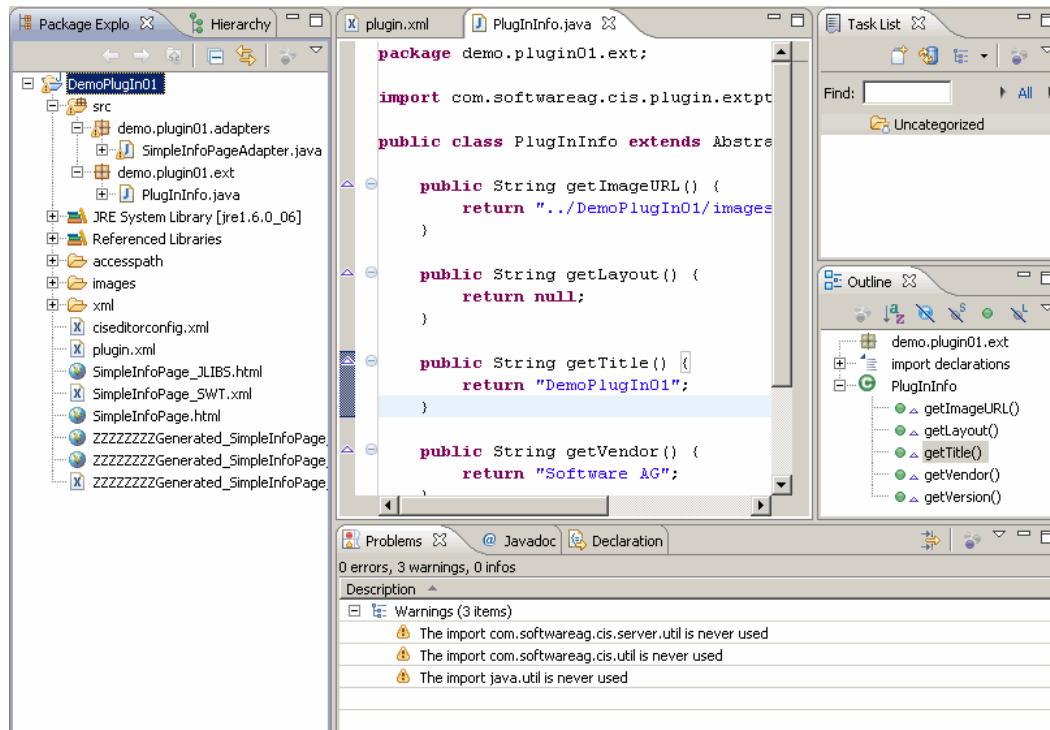
- c. Click **Save and Apply**.
- d. In the **Layout** view, select the tree node **page**.
- e. Change the `model` property of **page** from `DummyAdapter` to `demo.plugin01.adapters.SimpleInfoPageAdapter`.
- f. Click the diskette symbol to save the layout. The content of the **Preview** view changes and indicates an error now. This is normal and can be ignored.
- g. Click the **Tools** tab of the IDE and select **Code Assistant**. The look of the IDE changes and the generated code for the page adapter is visible on the right side.



You can apply the necessary changes for the adapter class using the IDE. A easier way is doing this inside the already existing eclipse project. Click the diskette icon again to save the adapter source code in the Java source directory of the CentraSite Control plug-in DemoPlugIn01.

4. Close the IDE and return to the Eclipse environment.
5. Refresh the plug-in project
 - a. Select the folder DemoPlugIn01 and select **Refresh** from the context menu.
 - b. Expanding all folders that relate to the plug-in.

The eclipse project should look like as shown. The contents of subfolders accesspath and xml have been created by the IDE activities. There is a new package called demo.plugin01.adapters containing the class SimpleInfoPageAdapter. The adapter classes used for plug-ins to CentraSite Control should be derived from the class BaseAdapter rather than from the class Adapter as provided by Application Designer.



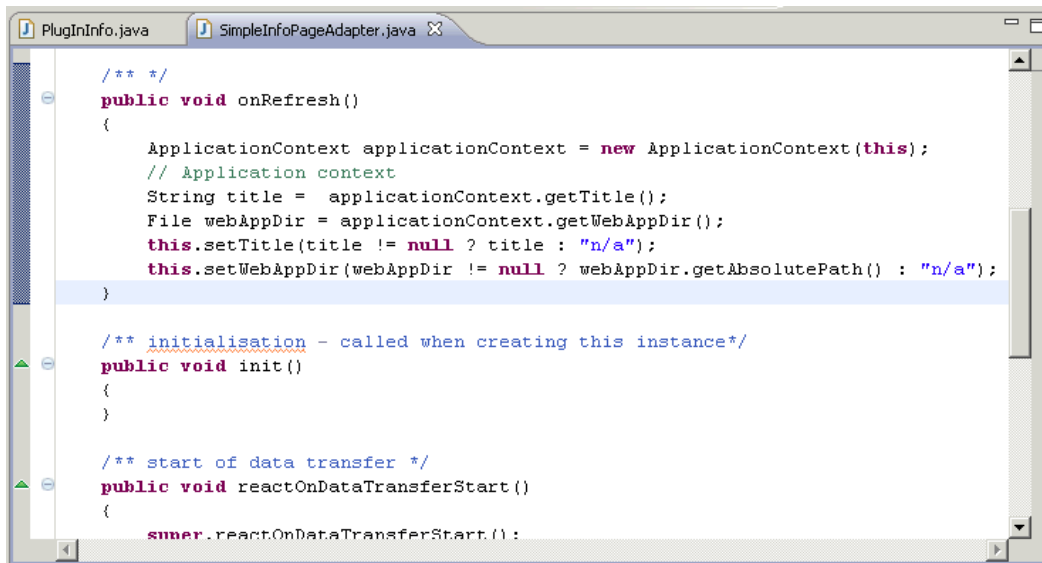
6. Apply application code to the adapter `SimpleInfoPageAdapter`.
 - a. Open the file `SimpleInfoPageAdapter.java` by double-clicking its node in the tree.
 - b. Type the following code inside the body of method `onRefresh`:

```
ApplicationContext applicationContext = new ApplicationContext(this);
// Application context
String title = applicationContext.getTitle();
File webAppDir = applicationContext.getWebAppDir();
this.setTitle(title != null ? title : "n/a");
this.setWebAppDir(webAppDir != null ? webAppDir.getAbsolutePath() : "n/a");
```

Some types are marked by the Java editor as unknown when you type the code.

- c. Press **Ctrl + Shift + O** to instruct the Java editor to add the necessary import statements automatically (for the missing class called `File` select `java.io.File` from the resulting dialog).
- d. Save the file

There should be no compilation errors.



7. Bring the user-defined layout inside CentraSite Control to the screen and extend CentraSite Control at another point in order to add a perspective

- a. Add the following XML element to the plug-in description file `plugin.xml` in the Eclipse environment and save it to inform the pluggable infrastructure that we are extending
- b. Add the following `requiredPlugin` XML element after the existing `requiredPlugin` XML element:

```
<requiredPlugin id="com.centrasite.control" />
```

- c. Add the following XML element to the already existing XML code:

```

<!-- Perspective -->
<extension point="com.softwareag.cis.plugin.perspective"
    id="DemoPlugIn01Perspective"
    class="demo.plugin01.ext.PlugInPerspective" >
</extension>

```

- d. Save the file `plugin.xml`.
8. The implementation of our new perspective requires a new class which implements the necessary interface:
 - a. In the context of `DemoPlugIn01/src`, create a new Java class called `PlugInPerspective` in package `demo.plugin01.ext`. Use class `com.softwareag.cis.plugin.extpt.util.AbstractPerspective` as the superclass.
 - b. Let method `getTitle()` return the string `DemoPlugIn01`.
 - c. Let the `getLogoImageUrl()` method return the path to our 24x24 icon (`../DemoPlugIn01/images/star-24x24.gif`).

d. Insert the methods.

```
public boolean hasTopicTree()
{
    return false;
}
```

and

```
public boolean supportsViews()
{
    return false;
}
```

The Java source should look like:

```
package demo.plugin01.ext;

import java.util.List;

import com.softwareag.cis.plugin.extpt.util.AbstractPerspective;
import com.softwareag.cis.plugin.extpt.util.WorkplaceContext;
import com.softwareag.cis.server.util.ICONLISTInfo;

public class PlugInPerspective extends AbstractPerspective
{

    public String getTitle()
    {
        return "DemoPlugIn01";
    }

    public String getImageURL()
    {
        return null;
    }

    public boolean hasTopicTree()
    {
        return false;
    }

    public boolean supportsViews()
    {
        return false;
    }

    public String getLogoImageURL()
    {
        return "../DemoPlugIn01/images/star-24x24.gif";
    }

    public ICONLISTInfo getToolbar()
    {
        return null;
    }

    public String getView()
    {
```

```
    return null;
}

public String getViewLabel()
{
    return null;
}

public List getViewValues()
{
    return null;
}

public String getWorkplaceDefaultLayout()
{
    return null;
}

public void setView(String arg0)
{
}

public void setWorkplaceContext(WorkplaceContext arg0)
{
}
}
```

e. Save and close the Java source file.

9. Extend CentraSite Control with a new topic.

- a. Add the following XML element to the plug-in description file `plugin.xml` in the Eclipse environment and save it.

```
<!-- Topic -->
<extension point="com.softwareag.cis.plugin.topic"
    id="DemoPlugIn01Topic"
    perspective="demo.plugin01.DemoPlugIn01Perspective"
    class="demo.plugin01.ext.PlugInTopic" >
</extension>
```

10. Implement the adapter class.

- a. In the context of `DemoPlugIn01/src`, create a new Java class called `PlugInTopicAdapter` in package `demo.plugin01.ext.adapters`. Use class `com.centrasite.control.adapters.TopicAdapter` as the superclass. Do not inherit abstract classes here.
- b. Add a public default constructor to the class. The Java source should look like:

```
package demo.plugin01.ext.adapters;

import com.centrasite.control.adapters.TopicAdapter;
```

```
public class PlugInTopicAdapter extends TopicAdapter
{
    public PlugInTopicAdapter()
    {
    }
}
}
```

- c. Save and close the Java source file.

11. Extend class.

- a. In the context of DemoPlugIn01/src, create a Java class called `PlugInTopic`, in the package `demo.plugin01.ext`, using the superclass `com.centrasite.control.ext.util.BaseTopic`. Select **Inherited abstract methods**.
- b. Add a public default constructor that invokes the `super(int)` constructor to the source.

```
public PlugInTopic ()
{
    super(0);
}
```

- c. Let method `getTopicAdapterClass()` return `PlugInTopicAdapter.class`.
- d. Let method `getTitle()` return the string `DemoPlugIn01`.
- e. Add the following code to method `initTree`.

```
String title = "Simple Info Page";
String pageUrl = "../DemoPlugIn01/SimpleInfoPage.html";
String adapterClass = SimpleInfoPageAdapter.class.getName();
ActionContext actionContext = getTopicAdapter().getActionContext();
actionContext.showPage(pageUrl, title, adapterClass);
```

- f. Press **Ctrl+Shift+O** to resolve compilation problems.
- g. Save the Java source file and make sure that no compilation errors occur. After applying the changes, the Java source code for class `PlugInTopic` should look like:

```
package demo.plugin01.ext;

import com.centrasite.control.ActionContext;
import com.centrasite.control.Item;
import com.centrasite.control.ext.util.BaseTopic;

import demo.plugin01.adapters.SimpleInfoPageAdapter;
import demo.plugin01.ext.adapters.PlugInTopicAdapter;

public class PlugInTopic extends BaseTopic
{
    public PlugInTopic()
```

```

{
    super(0);
}

protected Class getTopicAdapterClass()
{
    return PlugInTopicAdapter.class;
}

protected void initTree() throws Exception
{
    String title = "Simple Info Page";
    String pageUrl = "../DemoPlugIn01/SimpleInfoPage.html";
    String adapterClass = SimpleInfoPageAdapter.class.getName();
    ActionContext actionContext = getTopicAdapter().getActionContext();
    actionContext.showPage(pageUrl, title, adapterClass);
}

public void refresh(Item arg0, int arg1)
{
}

public String getTitle()
{
    return "DemoPlugIn01";
}

public String getImageURL()
{
    return null;
}
}

```

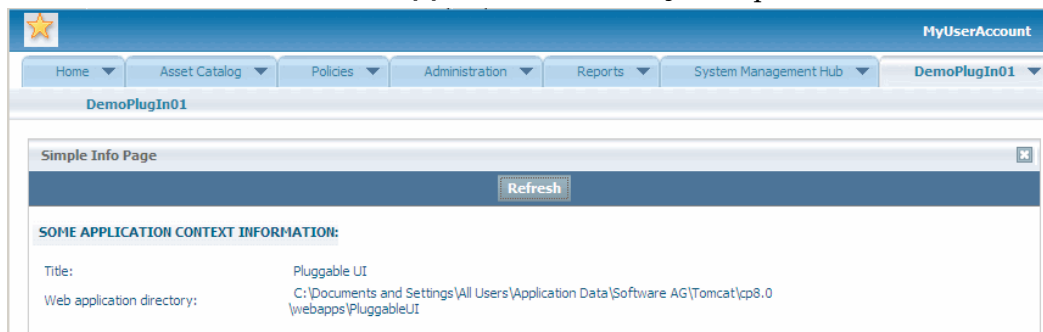
12. Restart the Software AG Runtime service.

13. Open CentraSite Control.

The navigation pane shows the new perspective **DemoPlugIn01** which has a topic entry **DemoPlugIn01**. Note that the `star-24x24.gif` graphic is visible in the header bar.

14. Click **Refresh** in the **Simple Info Page** display.

The values for **Title** and **Web application directory** are updated as shown.



Implementation of Computed Attributes and Profiles

This section describes implementing computed attributes and implementing computed profiles for CentraSite Business UI and CentraSite Control.

Implementing Computed Attributes

A computed attribute is defined as an implementation of the `ComputedAttribute` interface.

The implementation of a Java-based computed attribute includes a set of predefined `ComputedAttribute` and `ProfileAttribute` interfaces. These implementation interfaces are bundled in the `CentraSiteUtils.jar`, which is available in the `<CentraSiteInstall_Directory>/redist` folder.

Interfaces	Description
<code>ComputedAttribute</code>	<p>Declares the basic rendering methods for a computed attribute interface. It extends the <code>ProfileAttribute</code> interface.</p> <p><code>boolean: isUsed()</code>: returns <code>true</code> if this attribute is used in at least one asset instance of the corresponding type.</p> <p><code>Collection: setValue(Collection)</code>: sets the attribute value.</p>
<code>ProfileAttribute</code>	<p><code>void: init(Collection<CentraSiteRegistryObject>, Locale)</code></p> <p>The parameter <code>Collection</code> includes a single <code>CentraSiteRegistryObject</code> asset instance for which the attribute value is to be obtained. The parameter <code>Locale</code> provides the preferred locale of the logged in user.</p> <p><code>Collection: getValue()</code>: returns the actual set of values that are represented by this attribute for an asset instance.</p> <p>If the data is of type <code>String</code>, the method returns a collection of <code>Strings</code>. If the data is of type <code>Relationship</code>, the method returns a collection of <code>RegistryObjects</code>. If the data is of type <code>File</code>, the method returns a collection of <code>ExternalLinks</code>.</p> <p><code>String: getAttributeKey()</code></p> <p><code>AttributeDescriptor: getAttributeDescriptor()</code>: returns the definition of attribute.</p> <p><code>String: getName()</code>: returns the JAXR-based name of attribute.</p>
<code>AttributeDescriptor</code>	<p>Declares the standard properties (<code>isReadOnly()</code>, <code>is Required()</code>...) for an attribute.</p> <p><code>String: getDataType()</code>: returns the data type of attribute. It can be one of the supported standard types (<code>xs:...</code>). The constants are referred from the <code>CentraSiteSlotType</code> interface, which is available in the package</p>

Interfaces	Description
	<p><code>com.centrasite.jaxr.type</code> and bundled as a <code>CentraSiteJAXR-API.jar</code> in the CentraSite installation.</p> <p>If the computed attribute is of association or relationship type and returns the <code>RegistryObjects</code>, then the attribute data type is <code>Relationship</code>. If the computed attribute is of classification type and returns the <code>Concept</code> objects, then the attribute data type is <code>Classification</code>. If the computed attribute is of file type, then the attribute data type is <code>File</code>.</p> <p>Object: <code>getDefaultValue()</code>: returns the default attribute value.</p> <p>String[]: <code>getEnumValues()</code></p> <p>int: <code>getMaxLength()</code>: returns the maximum length of attribute.</p> <p>String: <code>getMaxOccurs()</code>: denotes if the attribute can have multiple occurrences. The options are: <code>ProfileAttribute.MAXOCCURS_1</code> and <code>ProfileAttribute.MAXOCCURS_UNBOUNDED</code>.</p> <p>String: <code>getMinOccurs()</code>: denotes whether the attribute is optional or mandatory. The options are: <code>ProfileAttribute.MINOCCURS_0</code> and <code>ProfileAttribute.MINOCCURS_1</code>.</p> <p>Object: <code>getNativeAttribute()</code>: returns the native attribute instance if it is not a primitive data type. Else, returns <code>null</code>.</p> <p>int: <code>getPrecision()</code>: returns the precision for a number attribute.</p> <p>String: <code>getUnitLabel()</code>: returns the slots' unit label.</p> <p>boolean: <code>hasDefaultValue()</code>: returns if the attribute has a default value.</p> <p>boolean: <code>isPrefix()</code>: returns <code>true</code> if the slots' unit label is a prefix, and <code>false</code> if the unit label is a suffix.</p> <p>boolean: <code>isReadOnly()</code>: denotes if the attribute is read-only.</p> <p>boolean: <code>isRequired()</code>: denotes if the attribute is required.</p>

Structure of Archive File

The plug-in uses the policy engine infrastructure, so it uses the structure of a Java policy (archive file structure and classloader). The archive file must contain the following folders and files:

Zip folder	Description
META-INF	This folder contains the <code>config.properties</code> file, which is the build file for the plug-in. This properties file contains an entry of the following format:

Zip folder	Description
	com.softwareag.centrasite.computed.attr.impl.class= com.sample.StringAttrImpl
lib	This folder contains the archive file with the source code examples, the plug-in's executor class and the external libraries.

Sample Code

The sample Java code for implementing a computed attribute in CentraSite is shipped in the `<CentraSiteInstall_Directory>/demos/ComputedAttribute/EndPointAttribute/src/com/softwareag/centrasite/demo/computedattribute/EndPointAttribute.java` folder.

A sample computed attribute is shipped in the `<CentraSiteInstall_Directory>/demos/ComputedAttribute/EndPointAttribute` folder.

Loading a Computed Attribute into an Asset Type Definition

After you have created an archive file that contains the attribute definition, you need to load the archive file into the asset type definition. You do this by starting the **Edit Asset Type** wizard for the appropriate existing asset type, or the **Add Asset Type** wizard for a new custom asset type and specifying in the wizard that you are defining a new computed attribute.

For procedures on how to load an archive file of a computed attribute into an asset type definition, see *CentraSite User's Guide*.

When you have loaded the archive file, the new attribute is displayed in the list of attributes that can be assigned to an asset type profile.

Implementing Computed Profiles for CentraSite Business UI

In the following sections, we demonstrate a custom profile named `SampleProfile` that illustrates how to write a computed profile. The sample extends the profile selection list for an asset type and presents a screen that prompts for the profile archive to be uploaded. After confirming the archive, the appropriate adapter classes are called.

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

Definition of Java-Based Computed Profile

A Java based computed profile has the following rendering mechanism:

- `WithUiRendering`: This dictates the user-defined rendering of the profile's attributes.
- `WithoutUiRendering`: This dictates the CentraSite's default rendering of the profile's attributes. This default rendering is based on the attribute's data type.

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, the Java-based build tool.

The following file system structure under the computed profile directory is assumed:

Name of File or Folder	Description
META-INF	This folder contains the <code>config.properties</code> file, which is the build file for the computed profile. This properties file contains an entry of the following format: <div><pre>com.softwareag.centrasite.computed.profil e.bui.impl.class=com.softwareag.centrasite.bui.profil e.server.SampleComputedProfileImpl</pre></div>
src	This folder that holds the Java source files.
lib	This folder contains the archive file with the source code examples, the plug-in's executor class and the external libraries.
html	This folder holds the html files that specify your computed profile window.
images	This folder holds the image files.
css	This folder holds the css files.
js	This folder holds the JavaScript codes.
build.xml	The Ant input file for building the destination files

The Ant file shown below, named `build.xml`, can be used to establish a custom computed profile.

```
<?xml version="1.0" encoding="utf-8"?>  
<project name="SampleProfile" default="all" basedir=".">  
  <property file="{basedir}/build.properties" />  
  <property name="src.dir" value="{basedir}/src" />  
  <property name="classes.dir" value="{basedir}/classes" />  
  <property name="build.output.dir" value="{basedir}/build_output" />  
  <path id="project.class.path">  
    <fileset dir="{gwt.home}">  
      <include name="gwt-dev.jar" />  
      <include name="gwt-user.jar" />  
      <include name="validation-api-1.0.0.GA.jar" />  
      <include name="validation-api-1.0.0.GA-sources.jar" />  
    </fileset>  
    <fileset dir="{centrasite.redist.dir}">  
      <include name="gf.jaxr-api-osgi.jar" />  
      <include name="INMConfiguration.jar" />  
    </fileset>  
    <fileset dir="{centrasite.rts.dir}">  
      <include name="CentraSiteLogicLayer-API.jar" />  
    </fileset>  
  </path>  
</project>
```

```

    <include name="CentraSiteLogicLayer-impl.jar" />
    <include name="CentraSiteUtils.jar" />
    <include name="gf.jaxr-api-osgi.jar" />
    <include name="CentraSiteJAXR-API.jar" />
  </fileset>
  <fileset dir="${others.dir}">
    <include name="gson-2.2.2.jar" />
  </fileset>
  <pathelement location="${src.dir}" />
</path>
<target name="all"
depends="clean, compile, compile-to-javascript,create-jar,create-zip" />
<target name="clean">
  <delete dir="${basedir}/gwt-unitCache" />
  <delete dir="${build.output.dir}" />
  <delete dir="${classes.dir}" />
  <delete dir="${basedir}/html/SampleProfile" />
</target>
<target name="compile">
  <mkdir dir="${classes.dir}" />
  <javac srcdir="src" destdir="${classes.dir}" debug="${javac.debug}"
  debuglevel="${javac.debuglevel}" optimize="${javac.optimize}"
  deprecation="${javac.deprecation}" classpathref="project.class.path"
  failonerror="true" memoryMaximumSize="512m" fork="true" />
</target>
<target name="compile-to-javascript" depends="compile"
description="GWT compile to JavaScript">
  <mkdir dir="${build.output.dir}" />
  <java failonerror="true" fork="true"
  classname="com.google.gwt.dev.Compiler" classpathref="project.class.path">
    <arg value="-war" />
    <arg value="${build.output.dir}" />
    <arg value="com.softwareag.centrasite.bui.profile.SampleProfile" />
    <jvmarg value="-Xms1024m" />
  </java>
</target>
<target name="create-jar" depends="compile"
description="create the jar by including the domain and server">
  <jar destfile="classes/lib/SampleProfile.jar">
    <fileset dir="classes"
    excludes="**/client/** **/junit/** **/test/** **/lib/**" />
    <manifest>
      <section name="com/softwareag/centrasite/bui/profile/server">
        <attribute name="Implementation-Title"
        value="Rule for SampleProfile profile assertion" />
        <attribute name="Implementation-Version"
        value="${component.full.version}.${working.build.number}" />
        <attribute name="Implementation-Vendor" value="Software AG" />
      </section>
    </manifest>
  </jar>
</target>
<target name="create-zip" depends="compile,compile-to-javascript,create-jar"
description="Creating the computed profile zip">
  <delete dir="html/SampleProfile" />
  <copydir src="${build.output.dir}/SampleProfile"
  dest="html/SampleProfile" />
  <zip update="yes" basedir="." includes="META-INF/**"
  destfile="${build.output.dir}/SampleProfile.zip" />
  <zip update="yes" basedir="." includes="html/**, css/**, js/**"

```

```
    destfile="${build.output.dir}/SampleProfile.zip" />
    <zip update="yes" basedir="classes" includes="lib/**"
    destfile="${build.output.dir}/SampleProfile.zip" />
  </target>
</project>
```

The classpath for the build step must refer to all JAR files contained in the `redist` folder of the CentraSite installation. Add these JAR files to the build path of your java project also.

Implementation Guidelines for Computed Profile

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.

`src\com\softwareag\centrasite\bui\profile\server\SampleComputedProfileImpl.java`

```
public class SampleComputedProfileImpl implements BUIProfile {
    private static final String VIEW_PAGE_URL = "html/SampleProfile.html";
    private static final String EDIT_PAGE_URL = "html/EditSampleProfile.html";

    private CentraSiteRegistryObject csro;
    private CentraSiteSession session;
    private Locale locale;

    @Override
    public boolean canRenderUI() {
        return true;
    }
    @Override
    public Collection<ProfileAttribute> getProfileAttributes() {
        return null;
    }
    @Override
    public void init(Collection registryObjects, Locale locale) {
        this.locale = locale;
        if (registryObjects == null || registryObjects.isEmpty()) {
            return;
        }

        Iterator iterator = registryObjects.iterator();
        while (iterator.hasNext()) {
            Object element = iterator.next();
            if (element instanceof CentraSiteRegistryObject) {
                csro = (CentraSiteRegistryObject) element;
            } else if (element instanceof CentraSiteSession) {
                this.session = (CentraSiteSession) element;
            }
        }
    }
    @Override
    public Collection<CentraSiteRegistryObject> updateAsset() {
        return null;
    }
    @Override
    public Collection<CentraSiteRegistryObject> computeProfileData(String arg0)
        throws Exception {
        return null;
    }
}
```

```

@Override
public String getProfileDataAsJson() throws Exception {
    ArrayList<ComputedInfo> computedInfos = new ArrayList<ComputedInfo>(2);
    computedInfos.add(new ComputedInfo("one", "One"));
    computedInfos.add(new ComputedInfo("two", "Two"));

    Gson gson = new Gson();
    return gson.toJson(computedInfos);
}

@Override
public String getViewPageURL() {
    return VIEW_PAGE_URL;
}

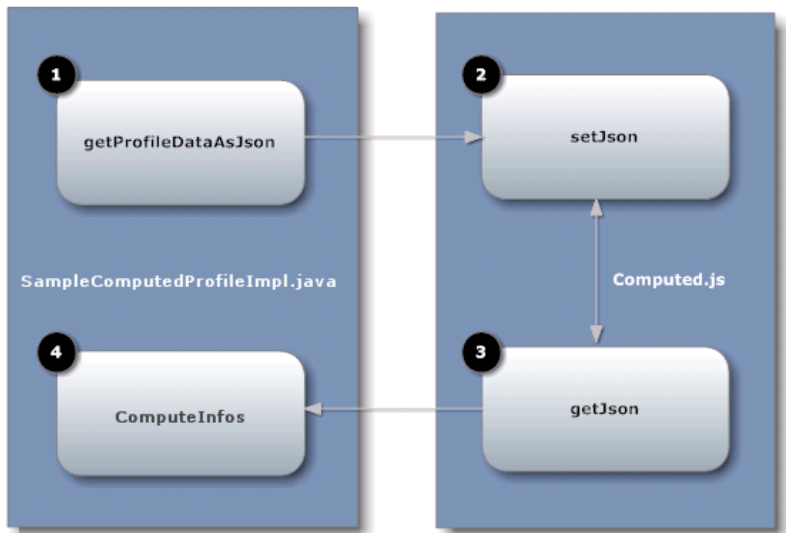
@Override
public String getEditPageURL() {
    return EDIT_PAGE_URL;
}
}

```

The `SampleComputedProfileImpl` class implements the interface `BUIProfile`, which declares the basic rendering methods specific to the CentraSite Business UI.

Implementations	Description
<code>boolean canRenderUI()</code>	Determines whether the rendering is based on the UI (true) or on the triples associated with the profile (false).
<code>Collection<ProfileAttribute> getProfileAttributes()</code>	Returns a collection of <code>ProfileAttribute</code> and would be called only when <code>canRenderUI()</code> returns true.
<code>void init(Collection registryObjects, Locale locale)</code>	With <code>CentraSiteRegistryObject</code> as a parameter where the necessary implementation is done and <code>updateAsset()</code> which would return a collection of registry object serves as a save hook.
<code>Collection<CentraSiteRegistryObject> updateAsset()</code>	Returns a collection of <code>CentraSiteRegistryObject</code> .
<code>String getEditPageURL()</code>	Returns URL of the edit page of computed profile and would be called only when <code>canRenderUI()</code> returns true.
<code>String getViewPageURL()</code>	Returns URL of the view page of computed profile and would be called only when <code>canRenderUI()</code> returns true.
<code>String getProfileDataAsJson()</code>	Returns a collection of profile data as JSON-formatted string.
<code>Collection computeProfileData(String userInputsAsJSON)</code>	Sets a collection of profile data as JSON-formatted string.

The following diagram describes the main methods on each of the two Java source files `SampleComputedProfileImpl.java` and `computed.js` and describes the type of functions that they serve.



#	Description
1	<p>The <code>getProfileDataAsJson</code> method returns a collection of profile data as JSON-formatted string and would be called only when <code>canRenderUI()</code> returns true.</p> <p>This method internally invokes the <code>setJson</code> function when trying to view or edit the computed profile asset.</p>
2	<p>The <code>setJson</code> method sets the JSON-formatted data representing the computed profile and would be called only when <code>canRenderUI()</code> returns true.</p> <p>This method is invoked when trying to view or edit the computed profile asset.</p>
3	<p>The <code>getJson</code> method retrieves the JSON-formatted data representing the computed profile using a HTTP GET request and would be called only when <code>canRenderUI()</code> returns true.</p> <p>This method is invoked when trying to save the updated asset.</p>
4	<p>The <code>computedInfos</code> method stores a collection of profile data as JSON-formatted string and would be called only when <code>canRenderUI()</code> returns true.</p> <p>This method is internally invoked by the <code>getJson</code> function when trying to save the updated asset.</p>

Here is the frame of the computed profile implementations:

`\js\computed.js`

```

/**
Function to resize the current profile frame
  
```



```

/**/
var resize = function() {
    parent.resizeFrame(profileId);
}
/**
    Function to validate the given input
**/
var validate = function() {
    return true;
}
/**
    Main function which is triggered from the computed profile
    infrastructure. Implementaion should be called from the function.
**/
var setJson = function(profileId, json, isView) {
    window.jsonData = json;
    window.sampleProfileId = profileId;

    console.log("isView = " + isView)

    try {
        if (isView) {
            renderSampleProfile();
        } else {
            editSampleProfile();
        }
    } catch(err) {
    }
}
/**
    Function to to indicate whether the current profile is modified or not
    Custom implementation can be possible here
**/
var isModified = function() {
    return false;
}

```

As mentioned above, in order to present a user-defined computed profile in the asset details page, HTML files (viewSampleProfile.html and editSampleProfile.html) that describe the GUI (in view or edit mode) must be located in the html directory.

\\html\\SampleProfile.html

```

<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <!-- -->
    <!-- Consider inlining CSS to reduce the number of requested files -->
    <!-- -->
    <!--<link type="text/css" rel="stylesheet" href="SampleProfile.css">-->
    <script type="text/javascript" language="javascript"
        src="SampleProfile/SampleProfile.nocache.js">
    </script>
    <script src="../js/computed.js"></script>
</head>

```

The HTML file specifies the Java source files that are user-defined to render the asset details page in the appropriate view or edit mode.

\\html\\EditSampleProfile.html

```
<head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8">
  <!--
  <!-- Consider inlining CSS to reduce the number of requested files -->
  <!--
  <!--<link type="text/css" rel="stylesheet" href="SampleProfile.css">-->
  <script type="text/javascript" language="javascript"
    src="SampleProfile/SampleProfile.nocache.js"></script>
  <script src="../js/computed.js"></script>
</head>
```

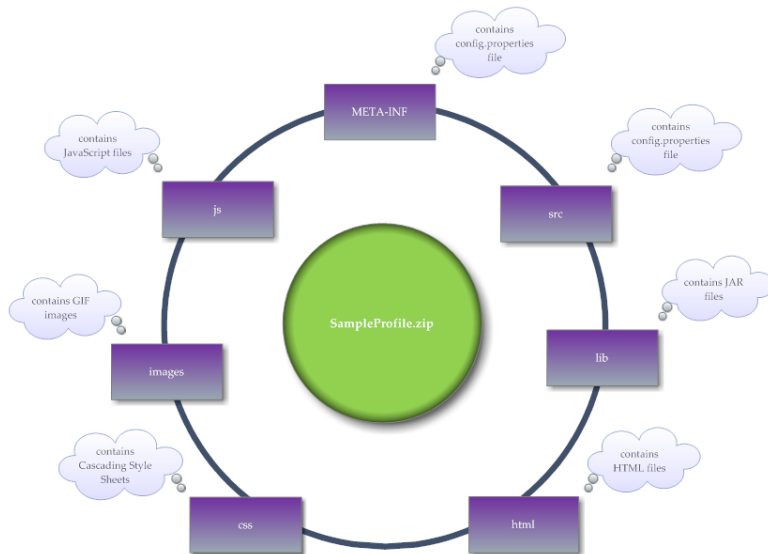
Assuming that you have set up all the Java files correctly in the directories, you should be able to build with the command:

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

This creates the profile specific JAR file in the subdirectory `lib` and archives the necessary profile files into the file `SampleProfile.zip`.

Structure of Archive File

The archive file must contain the following folders and files:



Sample Code

Your CentraSite installation contains a sample computed profile, `SampleProfile` (in the `demoss` folder) that you can use to create an archive file for the computed profile specific to the CentraSite Business UI.

Load a Computed Profile into an Asset Type Definition

After you have created an archive file that contains the profile definition, you need to load the archive file into the asset type definition. You do this by starting the **Edit Asset Type** wizard for

the appropriate asset type and specifying in the wizard that you are defining a new computed profile.

For details about how to load the archive file of a computed profile into an asset type definition, see *CentraSite User's Guide*.

When you have loaded the archive file, the new profile is displayed in the detail page of all assets of the asset type.

Implementing Computed Profiles for CentraSite Control

The following sections describe how to define a computed profile for the CentraSite Control UI.

Definition of Java-Based Computed Profile

A Java based computed profile has the following rendering mechanism:

- **WithUiRendering:** This dictates the user-defined rendering of the profile's attributes.
- **WithoutUiRendering:** This dictates the CentraSite's default rendering of the profile's attributes. This default rendering is based on the attribute's data type.

Implementation Guidelines for Java-Based Computed Profiles

This section describes the Java interfaces and methods that you need to implement for a computed profile.

Interfaces	Description
ComputedProfile	<p>This interface declares basic rendering methods for the user interface.</p> <p>void: <code>init(java.lang.Object, Locale)</code>: with <code>CentraSiteRegistryObject</code> as a parameter where the necessary implementation is done and <code>updateAsset()</code> which would return a collection of registry object serves as a save hook.</p> <p>boolean: <code>canRenderUI()</code>: This determines whether the rendering is based on the UI (true) or on the triples associated with the profile (false).</p> <p>Collection: <code>getAttributes()</code>: returns a collection of <code>ProfileAttribute</code> and would be called only when <code>canRenderUI()</code> returns true.</p> <p>Collection: <code>updateAsset()</code>: returns a collection of <code>CentraSiteRegistryObject</code> and would be called only when <code>canRenderUI()</code> returns true.</p>
WebUIProfile	This interface is specific to the rendering for CentraSite Control.

Interfaces	Description
	<code>java.lang.Object: createProfileContent()</code> : create the profile contents with XML content (in compliance with Application Designer).
EclipseProfile	<p>This interface is specific to the rendering for Eclipse Designer.</p> <p><code>createFormContent()</code>: create the profile contents on the supplied <code>IManagedForm</code> (in compliance with Eclipse Designer SWT/JFace library).</p>
BUIProfile	<p>This interface is specific to the rendering for CentraSite Business UI.</p> <p><code>getEditPageURL()</code>: returns URL of the edit page of computed profile and would be called only when <code>canRenderUI()</code> returns true.</p> <p><code>getViewPageURL()</code>: returns URL of the view page of computed profile and would be called only when <code>canRenderUI()</code> returns true.</p> <p><code>getProfileDataAsJson()</code>: returns a collection of profile data as JSON-formatted string.</p> <p><code>Collection computeProfileData (String userInputsAsJSON)</code>: sets a collection of profile data as JSON-formatted string.</p>
ProfileAttribute	<p>This interface deals with the standard UI rendering of the profile's attributes and allows defining the attributes as key value pairs. The rendering of each attribute is the standard rendering for the corresponding datatype of the attribute.</p> <p><code>AttributeDescriptor: getAttributeDescriptor()</code>: returns the descriptor of the attribute.</p> <p><code>String: getAttributeKey()</code>: returns the key of the attribute.</p> <p><code>String: getName()</code>: returns the name of the attribute.</p> <p><code>Collection: getValue()</code>: returns the value of the attribute.</p>
ComputedAttributeLine	<p>This interface deals with the user-defined UI rendering of the profile's attributes. The UI rendering of the attributes is determined by the coding of this interface.</p> <p><code>void: buildUI(StringBuilder layout)</code>: This method is responsible for rendering the layout definition.</p> <p><code>void: passivate()</code>: This method is responsible for storing the values back to the object.</p> <p><code>void: revert()</code>: This method is to revert the changes.</p>
AttributeDescriptor	<p>This interface deals with the standard properties (<code>isReadOnly()</code>, <code>isRequired()</code>...) for an attribute.</p>

Interfaces	Description
	String: getMinOccurs(): returns the minimum allowed occurrences of this attribute.
	String: getMaxOccurs(): returns the maximum allowed occurrences of this attribute.
	boolean: isRequired(): returns whether this attribute is required.
	boolean: isReadOnly(): returns whether this attribute is read-only.
	boolean: hasDefaultValue(): returns whether this attribute has a default value.
	Object: getDefaultValue(): returns the default value of this attribute.
	int: getMaxLength(): returns the maximum length of this attribute.
	String: getDataType(): returns the data type of this attribute.
	Object: getNativeAttribute(): returns the native attribute instance if the data type is not primitive, otherwise returns null.
	String[]: getEnumValues():
	boolean: isPrefix(): returns whether this slot's unit label is a prefix or suffix value. Return <code>true</code> if the unit label is a prefix, <code>false</code> for suffix.
	int getPrecision(): returns the precision for a number attribute.
	String: getUnitLabel(): returns this slot's unit label. The label may be null.

Structure of Archive File

The plug-in uses the policy engine infrastructure, so it uses the structure of a Java policy (zip file structure and classloader). The archive file must contain the following folders and files:

Zip folder	Description
META-INF	<p>This folder contains the <code>config.properties</code> file, which is the build file for the plug-in. This properties file contains an entry of the following format:</p> <pre>com.softwareag.centrasite.computed. profile.webui.impl.class= com.sample.MyProfileImpl</pre>
lib	This folder contains the archive file with the source code examples, the plug-in's executor class and the external libraries.

Sample Code

Your CentraSite installation contains two sample computed profiles (which is contained in demos folder) that you can use to create an archive file for the computed profile specific to CentraSite Control.

- NonPrimitiveDataTypeSamples
- SampleComputedProfile

Load a Computed Profile into an Asset Type Definition

After you have created an archive file that contains the profile definition, you need to load the archive file into the asset type definition. You do this by starting the **Edit Asset Type** wizard for the appropriate asset type and specifying in the wizard that you are defining a new computed profile.

For details about how to load the archive file of a computed profile into an asset type definition, see *CentraSite User's Guide*.

When you have loaded the archive file, the new profile is displayed in the detail page of all assets of the asset type.

Customizing CentraSite i18n Messages

CentraSite supports Internationalization (i18n) out of the box. With CentraSite's extensibility you can customize the default i18n messages that appear in the Business UI based on a user's language.

Understanding Message Bundles

To use different languages in CentraSite you create message bundle file for each different languages that you wish to render. Message bundles in CentraSite are located on `<CentraSiteInstall_Directory>/cast/cswebapps/BusinessUI/custom/resources` folder and are simple Java properties files.

Each bundle starts with the name `i18nresources` by convention and ends with the language (`_en`). By default, CentraSite looks in the `i18nresources_xx.properties` file for messages based on the user specific language.

Customizing i18n Messages

The i18n messages are stored in the form of key/value pairs. The key is the identifier used by CentraSite to retrieve the text, and the value is the actual text. CentraSite allows you to modify the default i18n messages to suit your needs.

After you decide which specific i18n message you want to customize, you must first fetch the key (unique identifier) from CentraSite's message database, and then manually update the i18n properties file.

Fetching Message Key

Pre-requisites:

To fetch a message key through the CentraSite Command Line Interface, you must have the CentraSite Administrator role.

CentraSite provides a Java tool named `i18NMessageFinder.jar` for this purpose.

➤ To fetch a message key

- Run the Java tool `i18NMessageFinder.jar`.

The syntax is of the format:

- `C:\SoftwareAG\CentraSite\utilities>CentraSiteToolbox.cmd I18NMessageFinder.jar -message <value of the message to be customized>`
- `C:\SoftwareAG\CentraSite\utilities>CentraSiteToolbox.cmd I18NMessageFinder.jar -file <config file> [-dump] [-locale <messages of this locale>]`

The input parameters are:

Parameter	Description
<code>-message</code>	The message for which key is to be found.
<code>-file</code>	Name of the configuration file containing all of the messages for which keys are to be found. For multiple messages, use a comma to separate the values.
<code>-dump</code>	Dumps all of the key-value pairs to be used for customization of messages. If <code>-file</code> parameter is specified, the key-value pairs are dumped in the given file. If <code>-file</code> parameter is not specified, the key-value pairs are displayed in the console.
<code>-locale</code>	Represents a language code of the form <code>xx</code> . The language code is always in lower case. The <code>-locale</code> parameter is used only if the <code>-dump</code> parameter is specified. Default value is <code>en</code> .

This process may take some time. The tool progress is reported to standard output.

Note:

Please make sure the combination of configuration parameters in the command syntax is valid. The invalid combinations are: `-message` and `-file` are mutually exclusive; `-message` and `-dump` are mutually exclusive.

The precedence of the configuration parameters for i18n customization is as follows:

- If you configure all of the parameters - `-message`, `-dump` and `-file` for customization, then the `-message` configuration takes precedence over the `-dump` and `-file` configurations.

- If you configure the parameters `-dump` and `-file` for customization, then the `-dump` configuration takes precedence over the `-file` configuration.

Examples (all in one line):

- Fetching message ID of a single message text

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteToolbox.cmd I18NMessageFinder.jar  
-message "Welcome to CentraSite Business UI"
```

The script generates a single message ID for the corresponding message text in the following format:

```
INMBU_STR_WELCOME_PAGE_MESSAGE
```

- Fetching multiple message IDs of the default locale in one execution of the tool:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteToolbox.cmd I18NMessageFinder.jar  
-dump
```

The script displays all of the messages that belong to the default English (en) locale in the console.

- Fetching multiple message IDs of the user-defined locale in one execution of the tool:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteToolbox.cmd I18NMessageFinder.jar  
-dump -locale ar
```

The script displays all of the messages that belong to the user-defined, for example, Arabic (ar) locale, in the console.

- Fetching multiple message IDs of multiple message texts in one execution of the tool:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteToolbox.cmd I18NMessageFinder.jar  
-dump - locale ar -file c:\tmp\customize-config.xml
```

Here, the parameter `customize-config.xml` specifies a plain-text file that contains the comma-separated message texts.

The script generates an `<config file>_results.properties` file that contains a list of the message IDs. Each line of the configuration file includes a message ID and the corresponding message text.

Changing Message Value

After you have identified the message key, you must then manually configure the `i18n` properties file in the following way:

1. Open the `i18nresources_xx.properties` file.

You can find the **`i18nresources_xx.properties`** file on `<CentraSiteInstall_Directory>/cast/cswebapps/BusinessUI/custom/resources`.

Note:

If you don't have an `i18nresources_xx.properties` file that suits your language, create one custom properties file that ends with the language you are interested. For example, `i18nresources_de.properties` for German language.

2. Add the message ID and the corresponding text you want to display in the CentraSite Business UI as follows:

`<ID of the message to be customized>=<text of the message to be customized>`

Example:

`INMBU_STR_WELCOME_PAGE_MESSAGE=Welcome to CentraSite`

3. Repeat the previous step for the required message IDs.
4. Save your configuration.
5. Restart Software AG Runtime.
6. Log in to the CentraSite Business UI to see your customized messages.

5 Custom Policy Actions

■ Custom Policy Actions	188
■ Computed Run-Time Actions	200

Custom Policy Actions

You can use a policy to execute a task that is not provided by a built-in action by creating a custom action. For example, a custom action can consist of a Java class that performs the required task, such as running a test, creating a required attribute, or logging an entry in an external database. You can insert custom actions into a policy.

Creating a custom action consists of the following high-level steps:

1. Create a custom *action category* to associate the action.
2. Create an *action template* to specify the scope of objects and events to which the action applies.
3. Specify the location of your custom *action rule* to be used in a design or change time policy for an action template.
4. Add *parameter templates* to define the parameters that serve as input parameters to the action.

There are two ways you can add custom actions:

- You can use CentraSite Control to create action categories and action templates, and create action rules using Java programs. You then upload the rules into action templates when you create the action templates, using CentraSite Control.
- You can create action categories, action templates, and action rules using Java programs. You then upload the rules scripts to action templates using JAXR-based calls in the action templates.

To create and manage action templates for design/change-time policies, you must belong to a role that includes the Manage System-wide Design/Change-Time Policies permission. By default, the following predefined roles include the Manage System-wide Design/Change-Time Policies permission:

- CentraSite Administrator
- Asset Type Administrator
- Operations Administrator

Action Categories

Action categories identify the action templates assigned to the category and the type of action those templates represent. CentraSite includes a set of predefined action categories and custom action categories.

Predefined Action Categories Installed with CentraSite

By default, the policy actions that are installed with CentraSite are grouped into these categories:

- ARIS Category
- Change-Time Category

- Collector Category
- Design-Time Category
- Global Category
- Handler Category
- Run-Time Category
- WS-I Compliance Category

Custom Action Categories

If you want to enforce policies using actions that are not provided by your CentraSite, you can create your own categories to define the custom actions. For example, a custom category can consist of user-defined actions to enforce policies. You can enforce policies of the custom actions just like you would enforce policies using the system (built-in) actions.

Action Templates

An action template specifies the object and event types to which the action applies. In addition, an action template for a design or change time policy contains your custom action rule, which fires when the action executes.

Types of Actions

CentraSite supports the following types of actions:

- *Manual Actions* are long-run processes that involve manual user intervention to complete the execution of the action. For example, Approval.

Note:

Although CentraSite allows you to use system-defined manual actions in creating policies, you cannot create a new manual action.

- *Axiomatic Actions* are simple actions used to configure parameters. No code is involved in the execution of axiomatic actions. Axiomatic actions are used in run-time policies.
- *Programmatic Actions* are usually executed by means of program code. Specifically, a programmatic action triggers an *action rule* when the action executes. You write an action rule as a Java class. Programmatic actions are used only in design/change-time policies. CentraSite provides sample action rules. You can upload action rules when you create a custom action template.

Supported Object Events

It is possible to create an action template whose scope encompasses any combination of object types and event types, however, not all combinations are enforceable. This is because, certain types of events do not occur for certain types of objects.

For example, a `PreStateChange` event occurs only on Assets, Policies, and Lifecycle Models. If you create a policy for a `PreStateChange` event on a User object, that policy does not execute because a `PreStateChange` event does not occur on a User object.

Parameter Templates

Parameter templates are a part of the action template. The parameter templates assigned to the action template serve as input parameters for the policy action at enforcement time. You can configure the required parameters of an action template in two ways:

- *Default values:* You can define parameters with multiple values and select one of these values as the default value. When the action template and its associated parameter template are used in a policy, you can restrict the policy action to use only the defined default values.
- *Blank values:* Alternatively, you can define parameters with blank values. When the action template and its associated parameter template are used in a policy, the policy action accepts any desired value.

Viewing Action Categories List

The **Action Templates** page displays the list of action categories and templates defined on your instance of CentraSite.

➤ To view the action categories and templates list

1. In CentraSite Control, go to **Policies > Action Templates**.
2. The action templates list provides the following information about a category or template:

Column	Description
Action Templates	Lists the action templates assigned to each category.
Description	Provides additional comments or descriptive information about an action template.
Type	Indicates whether an action category contains design/change-time or run-time action templates and whether an action template type is manual or programmatic.

Creating Custom Actions Using the CentraSite UI

You can create a custom action category and save it to CentraSite.

➤ To create a custom action category

1. In CentraSite Control, go to **Policies > Action Templates**.
2. Click **Add Action Category** available on the upper-right corner of the **Policy Information** panel.
3. In the **Add Action Category** dialog box, do the following:
 - a. Specify a name for the new custom category.

 An action category name need not be unique within the CentraSite Registry. However, to reduce ambiguity, you should avoid giving multiple action categories the same name.

 An action category name can contain any character (including spaces).
 - b. Select the type of template that the category contains (for example, Design/Change-Time, or Run-Time templates).
 - c. Click **OK**.

Adding an Action Template to a Custom Action Category

You can add an action template to a custom action category and save it to CentraSite.

» To add an action template to a custom action category

1. In CentraSite Control, go to **Policies > Action Templates**.
2. Click **Add Action Template**.
3. If a custom action category does not exist, the **Add Action Category** dialog box appears, prompting you to create a custom action category.

After you create a custom action category, the Add Action Template page appears and is described in the next step.

4. In the Add Action Template page, do the following:

In this field...	Do the following...
Category	Select the action category for which you want to add the action template.
Name	Type a name for the new action template. Follow these guidelines: <ul style="list-style-type: none"> ■ An action template name must be unique. ■ An action template name can contain any character (including spaces).

In this field...	Do the following...
Description	(Optional). Type a description for the new action template. This description appears when the user displays a list of action templates in the Policy Information panel.
Type	No action is necessary. By default, CentraSite sets the action type to Programmatic for a design-time or change-time action and Axiomatic for a run-time action.
Uploaded File	Click the Browse button and upload the action's rule (Java Zip file). Note: Alternatively, you can download the rules used by the system action templates, and modify them for use with the custom action template.

5. In the **Scope** panel, do the following:

In this field...	Specify...
Object Types	Select the type of objects to which this action template applies.
Event Types	Select the type of events to which this action template applies. Note: Not all event types are supported by all objects.

6. Click **Save**.

Upon saving the action template, CentraSite displays the Edit Action Template Detail page. You use this page's **Parameter Templates** profile to add parameter templates for this action.

Adding Parameter Template to Action Template

To complete the action template, you must define its input parameters.

> To add a parameter template to the action template

1. In CentraSite Control, go to **Policies > Action Templates**.
2. Select the action template for which you want to define parameter templates.
3. In the Edit Action Template Detail page, select the **Parameter Templates** profile.
4. Click **Add Parameter Template**.

5. Define the first parameter as follows:

In this field...	Do the following...
Name	Type a name for the new parameter template.
Type	Select a data type.
Default Value	<p>If you want to specify a default value, type a value in this field.</p> <p>If the selected data type is String, Number, or URL, you can specify one or multiple default values. You can specify <i>multiple</i> possible values from which to select as follows:</p> <ol style="list-style-type: none"> Select the Edit icon to the right of the Default Value field. In the Add Default Values dialog box, type a value and click Add. Repeat for as many values as you need. Click OK. In the Default Value field select from the drop-down list the value you want to use as the default value. <div style="background-color: #f0f0f0; padding: 10px; margin-top: 10px;"> <p>Note: If you would rather fill in required values when this template is used in a policy, leave this field blank.</p> </div>
Array	Select this check box if you want the data type as an array.
Required	Select this check box if you want the parameter template to be mandatory.

6. If you want to define additional parameters, click **Add Parameter Template** and repeat the previous step.
7. Click **Save** and then **Close**.

The parameter templates that you added appear under the **Parameter Templates** profile.

Creating Action Rules in a Java Class

You can create a rule in a Java class.

➤ To create a rule in a Java class

- Create a Java action executor class that implements the `com.softwareag.centrasite.policy.api.IActionExecutor` interface. To view the Javadoc for this interface, see the *CentraSite Java API Reference*.

Important:

The Java executor class must return an `AssertionResult` object that contains the completion code `ResultStatus.SUCCESS` (if the action was successful) or `ResultStatus.FAILURE` (if the action failed). There are other possible completion codes (for example, `ResultStatus.IN_PROCESS`), however, these codes are used by internal processes and *are not* intended to be returned by user-defined actions. Custom actions that you create must only return a completion code of `ResultStatus.SUCCESS` or `ResultStatus.FAILURE`.

2. Create the `<action_name>.zip` file by executing the ant command `"ant -f build_demos_customer.xml"`.

Example:

```
C:\SoftwareAG\CentraSite\demos\CustomActions\Java>ant -f build_demos_customer.xml
```

The `<action_name>.zip` file should include the following components:

- A folder named `lib`, which would contain a jar file with the action's executor class and the external libraries.
- A folder named `META-INF`, which would contain a property file named `assertion.properties`, which is the build file for the action. It includes an entry of the following format:

```
com.softwareag.centrasite.policy.rule.class=<fully_qualified_class_name>
```

3. Upload the `<action_name>.zip` file.

Uploading Action Rules to Action Templates

Before you upload a Java action rule, you must first create a `.zip` file.

➤ To upload an action rule

There are two ways you can upload an action rule (Java Zip file) to a custom action template:

- If you are uploading the action rule to a custom action template through CentraSite Control, you upload the Java Zip file when creating the template in the Add Action Template page.
- If you are uploading the action rule to a custom action template programmatically, you upload the Java Zip file using a JAXR-based call in the action template.

Viewing or Modifying an Action Category

You can use the **Edit Action Category** to examine and modify the properties of an action category. When viewing or changing the properties of an action category, keep the following points in mind:

- You cannot modify or delete the predefined categories that are installed with CentraSite.
- You can change any property of an custom category; however cannot modify the type.
- You can modify a category only after deactivating all the policies that use it.

- You can rename an action category at any time

➤ **To view or modify the properties of an action category**

1. In CentraSite Control, go to **Policies > Action Templates**.
2. In the **Policy information** panel, select the action category whose details you want to view or modify.
3. Examine or modify the category's properties on the **Edit Action Category** dialog box as appropriate.

Viewing or Modifying an Action Template

You can use the **Edit Action Template** to examine and modify the properties of an action template. When viewing or changing the properties of an action template, keep the following points in mind:

- You cannot modify or delete the predefined action templates that are installed with CentraSite.
- You can change any property of an action template; however cannot modify the type.
- You can modify an action template only after deactivating all the policies that use it.

➤ **To view or modify the properties of an action template**

1. In CentraSite Control, go to **Policies > Action Templates**.
2. In the **Policy information** panel, select the action template whose details you want to view or modify.
3. Examine or modify the template's properties on the Edit Action Template page as appropriate.
4. Select the **Scope** profile. View or modify the object types and event types to which this action template applies as appropriate. To modify the list of object or event types, do the following:
 - a. Click the **Select** button beside the list of applicable object or event types.
 - b. Use the controls in the Select Object/Event Types dialog box to adjust the list.
 - c. Click **Save** to update the modification.

Note:

Not all event types are supported by all objects.

5. Select the **Parameter Templates** profile. View or modify the parameter fields as appropriate.

6. If you have modified a template's properties, click **Save**.

Downloading Rules from System Action Templates

You can download the rules associated with CentraSite system action templates.

➤ To download a rule

1. In CentraSite Control, go to **Policies > Action Templates**.
2. Locate the action template whose rule you want to download and select its name.
3. Select the .zip file in the **Uploaded File** field and then download the rule.

The structure of the zip file created by the download feature is as follows:

- A folder named lib, which contains a jar file with the action's executor class and the external libraries.
- A folder named META-INF, which contains a property file named `assertion.properties`, which is the build file for the action. It includes an entry of the following format:

```
com.softwareag.centrasite.policy.rule.class=<fully_qualified_class_name>
```

Delete Custom Action Categories and Templates

When you delete custom action categories, action templates and parameter templates, you must delete these items in the below order:

1. Parameter templates
2. Action templates
3. Action categories

Deleting a Parameter Template

Before you attempt to delete a parameter template, you must first delete all the policies consuming it.

➤ To delete a parameter template

1. In CentraSite Control, go to **Policies > Action Templates**.
2. Locate the custom action template whose parameter template you want to delete and select its name.

3. Select the **Parameter Templates** profile.
4. Select the parameter template and click **Delete**.

This temporarily revokes the selected parameter template from the action template.

5. Click **Save**.

Deleting a Custom Action Template

Before you attempt to delete a custom action template, you must first delete all the policies consuming it. Note that when you delete a custom action template, CentraSite also deletes all previous versions of the template.

➤ To delete a custom action template

1. In CentraSite Control, go to **Policies > Action Templates**.
2. Locate the custom action template that you want to delete.
3. Select the action template and click **Delete**.

Deleting a Custom Action Category

Before you attempt to delete a custom action category, you must first delete all the policies that are using the action templates under this category.

➤ To delete a custom action category

1. In CentraSite Control, go to **Policies > Action Templates**.
2. Locate the custom action category that you want to delete.
3. Select the action category and click **Delete**.

Versioning a Custom Action Template

If you want to modify a custom action template, you can create a new version of the existing template and modify the new version as required. When you create a new version of a custom action template, CentraSite creates an identical copy of the existing template; you can then modify the copy as required. (The new version of the custom action template gets its own copy of the Java Zip file.)

Note that CentraSite *does not* automatically apply the new custom action template to policies that use existing versions of the custom action. Policies that use existing versions of the action continue to use the versions that they have. If you want to apply the new version of the action to these

policies, you must edit the policies (or create new versions of them) and replace the old version of the action with the newer one.

Similarly, modifying the parameter definitions in a new version of an action template *does not* affect the parameter definitions in any of the existing policies that use the action. Parameter definitions are specific to a version of the template.

When you create a new version of a custom action, be aware that:

- You can only create a new version from the *latest version* of an action. For example, if an action already has versions 1.0, 2.0, and 3.0, CentraSite only allows you to create a new version of the action from version 3.0.
- The new version of the action is identical to the version from which you created it (except for the system-assigned version identifier, which is always incremented by one).
- CentraSite automatically establishes a relationship between the new version of the policy action and the previous version. CentraSite uses this relationship to identify versioned actions.
- You can only create new versions of *custom* actions that exist on your instance of CentraSite (that is, actions that you have added to CentraSite). You cannot create new versions of the predefined actions that are installed with CentraSite.

➤ To version a custom action

1. In CentraSite Control, go to **Policies > Action Templates** to display the list of action templates.
2. Locate the most recent version of the custom action for which you want to create a new version.
3. From the context menu for the custom action, click **Create New Version**.
4. Modify the new version of the custom action as required and save it.

Tip:

To make the new version of the custom action distinguishable from earlier versions, consider appending the version number to the name of the custom action.

Sample Java Action: Enforce Unique Asset Names

You can use the sample Java class action rule `uniquenamechecker` that is present in the CentraSite installation to create a custom action, Unique Name Checker. This action ensures that the name and version combination of a newly-created asset is unique within the CentraSite catalog.

Note:

If the name and version combination of a newly-created asset is not unique, then the action returns a Failure message, and does not create the new asset in CentraSite registry.

To create the custom action, you have to:

1. Create a custom action category.

2. Create a custom action template, to which you upload the sample Java rule.
3. Create a Design/Change-Time policy and add the custom action template to it.

➤ **To create and test the custom action Enforce Unique Asset Names**

1. In CentraSite Control, go to **Policies > Action Templates**.
2. Click the **Add Action Category** button.
3. In the **Add Action Category** dialog box, do the following:
 - a. Specify a name for the new custom category, for example `My Custom Actions`. An action category name can contain any character (including spaces).
 - b. Select **Design/Change-Time** as the action category type.
 - c. Click **OK**.

The action category that you created appears as a custom category next to an icon in the Policy Information panel.

4. Click **Add Action Template**.
5. In the Add Action Template page, specify the following information:

In this field...	Do the following...
Category	Select the custom action category you just created.
Name	Type the name <code>Enforce Unique Asset Names</code> for the new action template.
Description	(Optional). Type a description for the new action template. For example: <code>Ensures that asset names are unique.</code>
Uploaded File	Click Browse and upload the Java class action rule: <code><CentraSiteInstall_Directory>\demos\CustomActions\Java\uniquenamechecker.zip</code>

6. In the **Scope** panel, specify the following information:

In this field...	Do the following...
Object Types	Select Service as the type of object to which this action template applies.

In this field...	Do the following...
Event Types	Select Pre-Create as the type of event to which this action template applies.

7. Click **Save**.

The Edit Action Template Detail page is displayed.

8. Create a policy and add the sample action to the policy as follows:
- In CentraSite Control, go to **Policies > Design/Change Time**.
 - Click **Add Policy**.
 - In the **Policy Information** panel, type a name for the new policy, for example, Ensure Unique Asset Names Policy. A policy name can contain any character (including spaces).
 - In the **Scope** panel, specify the object and event types to which the policy applies as follows:
 - In the **Object Types** field, select **Service** as the type of object to which this policy applies.
 - In the **Event Types** field, select **Pre-Create** as the type of event to which this policy applies.
 - In the **Organization** field, select your organization name as the organization to which this policy belongs (and to whose objects the policy is applied).
 - Click **Next**.
 - From the **Available Actions** list, select the custom action **Enforce Unique Asset Names** action that you created.
 - Click **Finish** to save the new policy.

The Design/Change-Time Policy Details page is displayed.

- h. Activate the policy by clicking the **Change State** button and selecting the **Productive** state.

Computed Run-Time Actions

CentraSite Business UI offers you the possibility to add computed runtime actions into the policy workflow; this gives you the option to define your own runtime action, which means that you can implement your own algorithms for representing the action's user interface.

Computed runtime actions let you create your own layout by using a UI Rendering Concept. You can also specify your own rendering logic to display the computed values. You could, for example, create a custom display of the attribute as a drop down or a radio button.

A computed runtime action can be implemented using the GWT framework. For a computed runtime action, you create an archive file that contains the plug-in definition, and you load the archive file in the CentraSite BUIExtension folder.

Writing Your Own Computed Runtime Action

A computed runtime action can be implemented as a plug-in. The prepared plug-in is a collection of files in a specific directory structure. After implementing the plug-in, the files are copied into the CentraSiteBUIExtension folder in `<CentraSiteInstall_Directory>\demos` directory.

In the following sections, we demonstrate a sample framework named `MyComputedRuntimeAction` that illustrates how a custom computed runtime action may be set up.

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

The Build Environment

This section explains the build environment for generating the files that are used for the GUI and for compiling the necessary Java source files. It assumes the use of Ant, the Java-based build tool.

The following file system structure under the computed runtime action directory is assumed:

Name of File or Folder	Description
src	This folder that holds the Java source files.
lib	This folder contains the archive file, plug-in's executor class and the external libraries.
build.xml	The Ant input file for building the destination files

The Ant file, `build.xml` can be used to establish a custom computed profile.

The classpath for the build step must refer to all JAR files contained in the `redist` folder of the CentraSite installation. Add these JAR files to the build path of your java project also.

Implementation for Computed Action UI

`src\com\softwareag\centrasite\bui\extension\client\runtime\action\MyComputedRuntimeActionWidget.java`

```
public class MyComputedRuntimeActionWidget extends Composite {
    private PolicyActionJSO policyActionJso = null;
    private TextBox valueBox = null;
    private static final String WARNING_CSS = "loginTextBoxErrorBorder";

    public MyComputedRuntimeActionWidget(String policyActionJson) {
        FlowPanel container = new FlowPanel();
```

```

        initWidget(container);

        policyActionJso = getPolicyActionJso(policyActionJso);
        if (policyActionJso == null) {
            Label helloLabel = new Label("The JSON content is empty");
            container.add(helloLabel);
            return;
        }

        //Render widgets
        container.add(getParametersView(policyActionJso));
    }

    private Widget getParametersView(PolicyActionJSO policyActionJso) {
        FlowPanel parametersContainer = new FlowPanel();

        JsArray<ParameterJSO> parameters = policyActionJso.getParameters();
        if (parameters == null) {
            return parametersContainer;
        }

        for (int i = 0; i < parameters.length(); i++) {
            parametersContainer.add(getParameterView(parameters.get(i)));
        }

        return parametersContainer;
    }

    private Widget getParameterView(ParameterJSO parameterJso) {
        FlowPanel parameterContainer = new FlowPanel();
        Label nameLabel = new Label(parameterJso.getName());
        parameterContainer.add(nameLabel);

        valueBox = new TextBox();
        valueBox.setLayoutData(parameterJso.getId());

        String[] values = parameterJso.getValues();
        if (values != null && values.length > 0) {
            valueBox.setValue(values[0]);
        }

        parameterContainer.add(valueBox);
        return parameterContainer;
    }

    public static native PolicyActionJSO getPolicyActionJso(String json) /*-
        return eval('(' + json + ')');
    */-;

    public String getJson() {
        JsArray<ParameterJSO> parameters = policyActionJso.getParameters();
        ParameterJSO parameterJso = null;
        if (parameters != null && parameters.length() > 0) {
            parameterJso = parameters.get(0);

            String[] values = {valueBox.getValue()};
            parameterJso.setValues(values);
        }

        return policyActionJso.toJSON();
    }

    public boolean isValid() {
        String value = valueBox.getValue();
    }

```

```

        boolean isValid = (value != null && !"".equals(value));
        if (!isValid) {
            valueBox.addStyleName(WARNING_CSS);
        } else {
            valueBox.removeStyleName(WARNING_CSS);
        }

        return isValid;
    }
}

```

The `MyComputedRuntimeActionWidget` class extends the class `Composite`, which declares the basic rendering methods for the CentraSite Business user interface.

Implementations and Description

`MyComputedRuntimeActionWidget(String policyActionJson)`

Constructor dictates the user-defined rendering of the action's UI.

`getPolicyActionJson`

Returns the JSON object from the specified object.

`String getJson()`

Returns a JSON encoded string representing the action's parameters.

`boolean isValid()`

Enforces validation logic for the action's parameter values.

Implementation for Computed Action Parser

To implement your own computed runtime action with custom UI rendering, the parser (`MyComputedRuntimeActionParser.java`) must be located in the service directory. A parser is responsible for generating compressed JSON data from the given policy action instance, and creating a custom rendering of the action instance using the JSON data.

Here is the frame of the computed runtime action parser implementation:

src\com\softwareag\centrasite\bui\extension\service\ `MyComputedRuntimeActionParser.java`

```

public class MyComputedRuntimeActionParser extends
    BasePolicyActionExtensionParser {
    public MyComputedRuntimeActionParser(CentraSiteSession centraSiteSession,
        CentraSitePolicyActionTemplate actionTemplate,
        CentraSitePolicyActionInstance actionInstance) {
        super(centraSiteSession, actionTemplate, actionInstance);
    }
    @Override
    public CentraSitePolicyActionInstance getActionInstance(String json)
        throws CLLEException {
        Gson gson = new Gson();
        MyComputedRuntimeActionInfo = gson.fromJson(json,
            MyComputedRuntimeActionInfo.class);
    }
}

```

```

        if (actionInfo == null) {
            return null;
        }

        CentraSitePolicyActionInstance policyActionInstance = null;
        CentraSiteObjectManager objectManager =
            getCentraSiteSession().getCentraSiteObjectManager();

        if (actionInfo.isActionInstance()) {
            policyActionInstance = objectManager.getPolicyActionInstance(action
Info.getId());
        } else {
            policyActionInstance = objectManager.createPolicyActionInstance(act
ionInfo.getId());
        }

        if (policyActionInstance == null) {
            return null;
        }

        setParameterValues(policyActionInstance, actionInfo);
        return policyActionInstance;
    }

    private void setParameterValues(CentraSitePolicyActionInstance
        policyActionInstance, MyComputedRuntimeActionInfo actionInfo)
        throws CLLEException {
        List<MyComputedRuntimeParameterInfo> parameters = actionInfo.getParame
ters();
        if (parameters == null || parameters.isEmpty()) {
            return;
        }

        MyComputedRuntimeParameterInfo parameterInfo = parameters.get(0);
        Collection<Object> convertedParameterValues = new ArrayList<Object>();
        convertedParameterValues.addAll(parameterInfo.getValues());
        policyActionInstance.setAttributeValue(parameterInfo.getId(),
            convertedParameterValues);
    }

    @Override
    public String getJson() throws CLLEException {
        Gson gson = new Gson();

        MyComputedRuntimeActionInfo actionInfo = null;
        if (getActionInstance() != null) {
            CentraSitePolicyActionTemplate policyActionTemplate =
                getActionInstance().getCentraSitePolicyActionTemplate();
            actionInfo = new MyComputedRuntimeActionInfo(getActionInstance().g
etId(),
                policyActionTemplate.getName());
            actionInfo.setActionId(policyActionTemplate.getId());
            actionInfo.setIsActionInstance(true);
        } else if (getActionTemplate() != null) {
            actionInfo = new MyComputedRuntimeActionInfo(getActionTemplate().g
etId(),
                getActionTemplate().getName());
            actionInfo.setActionId(getActionTemplate().getId());
        }

        fillParameterInfos(getActionTemplate(), actionInfo);
        return (actionInfo != null ? gson.toJson(actionInfo) : null);
    }

```

```

    }
    private void fillParameterInfos(CentraSitePolicyActionTemplate
                                   actionTemplate,
                                   MyComputedRuntimeActionInfo actionInfo) throws CLLEException {
        if (actionTemplate == null) {
            return;
        }

        Collection<CentraSiteObjectAttribute> attributes = actionTemplate.getAttributes();
        if (attributes == null || attributes.isEmpty()) {
            return;
        }

        List<MyComputedRuntimeParameterInfo> parameters = new
            ArrayList<MyComputedRuntimeParameterInfo>(attributes.size());
        MyComputedRuntimeParameterInfo parameter = null;
        for (CentraSiteObjectAttribute attribute : attributes) {
            parameter = new MyComputedRuntimeParameterInfo(attribute.getName(),
                                                            attribute.getDisplayName());
            parameters.add(parameter);
        }

        actionInfo.setParameters(parameters);
    }
}

```

Computed Run-Time Action Plug-in

The following list shows the main methods on each of the two Java source files `MyComputedRuntimeActionWidget.java` and `MyComputedRuntimeActionParser.java` and describes the type of functions that they serve.

- The `getPolicyActionJson` method returns the JavaScript Object (JSO) from the given JSON-formatted string.
- The `getActionInstance` method returns a `CentraSitePolicyActionInstance` object from the given JSON-formatted string.
- The `String getJson` method returns a JSON-formatted string from the existing policy action instance.
- The boolean `isValid()` method enforces a validation logic for the user-defined rendering of the runtime action.

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

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

Activating the Computed Run-Time Action

After you define the computed action as a plug-in (extension point) with the above steps, enable the computed action in the Business UI customization file, **centrasite.xml**, in order to display the action in the policy accordion.

Important:

Remember that the action parameters defined in the configuration file are editable and cannot be protected.

> To activate the plug-in

1. Open the customization file, **centrasite.xml**, in a rich text editor.

You can find the **centrasite.xml** file on `<CentraSiteInstall_Directory>\cast\cswebapps\BusinessUI\custom\conf`.

2. Navigate to the property lines `<UIProperties> -> <Extensions> -> <PolicyActions>`
3. Append the property statement for your custom computed runtime action (MyComputedRuntimeAction) as below:

```
<PolicyActions>
  <PolicyAction id="uddi:44e3e2de-064c-432f-b67a-8fbca0fb04d6"
    class="com.softwareag.centrasite.bui.extension.service.
    MyComputedRuntimeActionParser" />
</PolicyActions>
```

wherein,

Parameter	Description
id	A unique identifier for the computed action. It uniquely distinguishes an action in the CentraSite registry. If you wish to reconfigure the action at a later stage, you identify the action using this id.
class	A parser implementation for the computed action.

4. Save and close the configuration file.
5. Restart Software AG Runtime.

Sample Computed Run-Time Action

Your CentraSite installation now contains few sample custom plug-in actions defined especially for the Business UI and available for use in the CentraSiteBUIExtension folder in

`<CentraSiteInstall_Directory>\demos` directory.

For more information about the implementation and usage of these sample custom plug-in actions, see the readme file included in the following directory:

`<CentraSiteInstall_Directory>\demos\CentraSiteBUIExtension`

6 Custom Reporting Searches

■ Writing Your Own Reporting Search	210
■ Displaying List of Reporting Searches	219
■ Obtaining Details of Reporting Search	220
■ Copying Reporting Search	221
■ Sample XQueries for Reporting Searches	222

Writing Your Own Reporting Search

CentraSite includes a set of predefined reporting searches that you can execute and capture design-time and/or run-time information of registry objects. In addition to the predefined reporting searches, you can create your own custom reporting search to execute and capture information on the registry objects that best suit your needs.

In CentraSite, you use queries in XQuery format to search and report for registry objects. You define custom queries and report your searches using XQuery modules and functions.

Reporting searches that are developed using XQuery modules and functions are applicable to CentraSite registry objects, such as reports and portlets in CentraSite Business UI.

In the following sections, we demonstrate a sample named, `TopXAssetsBasedonAPIKeys` that illustrates how a custom reporting search can be set up using the XQuery module and functions in CentraSite Business UI.

The sample extends to use the **Add New Portlet** dialog box which prompts for the data set you need to display in the new portlet. After confirming the data set, the corresponding search results are displayed in the newly created portlet `Top X Assets Based on API Keys`.

You may use this sample as a guideline, adapting it and renaming it to suit your individual requirements.

To add a custom reporting search in CentraSite Business UI, you must perform the following steps:

- Write XQuery Module
- Upload XQuery Module to CentraSite Registry Repository
- Write Saved Search XML
- Upload XML Search File from the Command Line
- Activate the Reporting Search

Writing XQuery Module and Functions

XQuery modules are libraries of user-defined functions. An XQuery module contains one or multiple functions that you require to retrieve specific data from the CentraSite registry using the XML query language. You can create a module of functions in one XQuery file and then use the functions defined in that XQuery module in any other XQuery by using the `import module` directive. The user-defined functions belonging to the same context can be bundled into modules. Thus based upon these functions the implementation of complex queries within that context becomes much easier.

An XQuery module is made up of a prolog and a body. In CentraSite XQuery, the XQuery prolog is a series of definitions and functions that together create the required environment for query processing.

The XQuery prolog includes namespace declarations. The XQuery body is made up of a sequence of expressions that specify the intended query result.

Example:

```
module namespace au="http://namespaces.CentraSite.com/modules/reports/apiusage";
import module namespace cs=
    "http://namespaces.CentraSite.com/Schema/jarx";
import module namespace csdt=
    "http://namespaces.CentraSite.com/modules/datetetime";
import module namespace util=
    "http://namespaces.CentraSite.com/jaxr/contants";
import module namespace util=
    "http://namespaces.CentraSite.com/modules/util";
import module namespace cs1=
    "http://namespaces.CentraSite.com/Schema";
```

The following XQuery is specified against the `getAPIsWithAPIKeys` declaration of integer type. The query retrieves the total number of APIs that use an API key to authenticate clients.

```
declare function au:getAPIsWithAPIKeys($noOfAssets as xs:integer) as node()*
{
  (
    let $ulp := csdt:getUserLocalePreferences()
    for $assetKey in au:getUniqueAPIswithAPIKeys()
    let $oi := collection("CentraSite")/cs:objectInfo[@v3key = @assetKey]
    let $apiKeysCount := au:getNumberOfAPIKeys($assetKey)
    order by $apiKeysCount descending
    return
    <result>
    <assetKey>{$assetKey}</assetKey>
    <assetName>{if(empty($oi/cs:name)) then ()
    else data(cs:localString($oi/cs:name, $ulp))}</assetName>
    <assetDesc>{if(empty($oi/cs:description)) then ()
    else data(cs:localString($oi/cs:description, $ulp))}</assetDesc>
    <apiKeysCountForAsset>{$apiKeysCount}</apiKeysCountForAsset>
    </results>
  )[position() <= $noOfAssets]
}
```

Note the following from the previous query:

- The XQuery prolog includes a namespace prefix (`cs`) declaration, namespace `cs="http://namespaces.CentraSite.com/Schema/jarx";`.
- The `declare namespace` keyword defines a namespace prefix (`cs`) that is used later in the query body.
- `let $oi := collection("CentraSite")/cs:objectInfo[@v3key = @assetKey]` is the query body.

CentraSite mandates XQuery modules and functions, especially for reports, when specifying data sets that are based on the CentraSite XQuery Data Source. The XQuery modules and functions help integrate the following functionality into the BIRT reports:

- Identify data sources that connect your reports to the CentraSite registry.
- Identify data sets that specify the data to retrieve from the CentraSite registry.

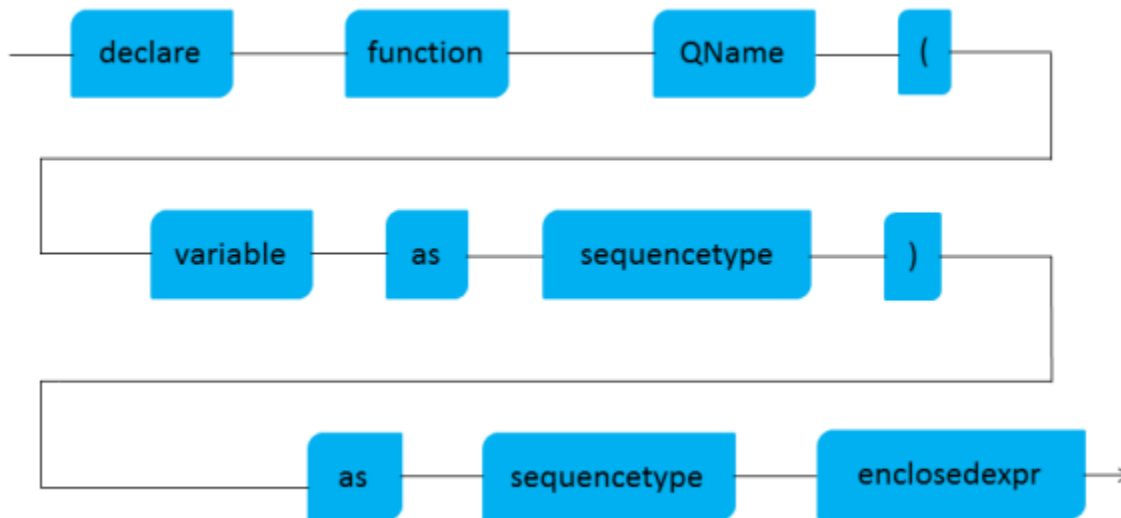
The XQuery functions must be written in the XQuery syntactical and semantical requirements of CentraSite, to specify the data to be retrieved from the registry and to be included in the data set,

and return a list of registry objects as its result; therefore, you must have a good knowledge of XQuery language.

The following query retrieves all registry objects of type Organization from the CentraSite registry.

```
declare namespace cs="http://namespaces.CentraSite.com/Schema/jaxr";
for $ro in collection("CentraSite")/cs:organization return $ro
```

The syntax for user-defined functions, which are declared in the XQuery prolog, is as follows:



The syntax diagram tells us that a function declaration starts with the two words `declare function` followed by a `QName`. The `QName` (=qualified name) must be in a namespace, that is, either a default function namespace declaration must be in effect or the `QName` must use a prefix. When using a prefix, a namespace binding declaration can be avoided by using the prefix `local`.

The function's name is followed by a possibly empty comma-separated list of variables that is enclosed in parentheses. Each variable may optionally be accompanied by a type specification. Also the result type of a function may optionally be specified. The term `enclosedexpr` is an expression enclosed in curly braces that contains what the function returns if called or rather what the function call expression evaluates to (in functional language speak).

Thus the simplest query containing (and using) a function declaration in our sample is:

```
declare function au:getAPIsWithAPIKeys($noOfAssets as xs:integer) as node()*

```

This query returns the total number of APIs that use an API key to authenticate clients.

Once you have the user-defined XQuery function `getAPIsWithAPIKeys()` defined in the above XQuery format, you must enclose the function in an XQuery module. In our sample, we call this module as `apiusage`.

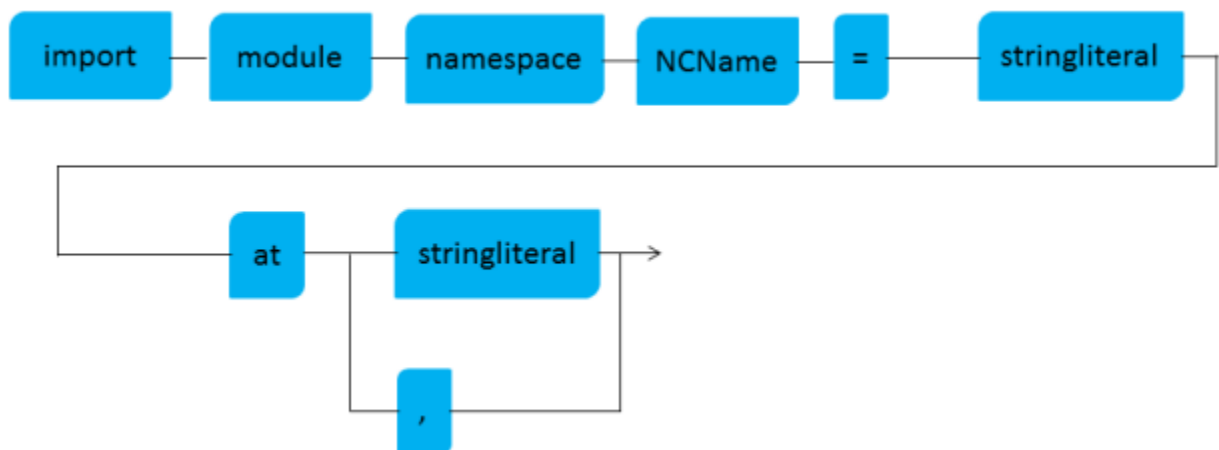
To enable queries to use user-defined functions without having to define them in every query, CentraSite XQuery allows for so called library modules. The declaration of such a library module is as follows:



A library module begins with the words `module namespace` followed by an `NCName` (a no-colon-name) that is used as a namespace prefix throughout the subsequent declarations. This is followed by an `=` sign and a string literal that specifies the namespace identifying the following declarations. This namespace specifier should be a URL that reflects the purpose of the module's content. The final semicolon (`;`) is optional due to compatibility reasons.

CentraSite stores modules in the database (Registry Repository) itself. After you create the required XQuery library module in to an XQuery file, you must upload it to the CentraSite Registry Repository.

A main module refers to library modules by importing the namespace that identifies the module's contents, that is, the module's target namespace as such:



A library module may be imported into a main module (or just as well into another library module) by writing `import module namespace` followed by a binding that binds a prefix to the library module's target namespace. The imported module's contents are then accessible using this prefix. The ability to bundle functions into modules can be exploited to make queries much more concise.

Thus the simplest way of importing a module in our sample is:

```
import module namespace cs="http://namespaces.CentraSite.com/Schema/jarx";
```

The final XQuery module would look like the following:

Uploading XQuery Module through CentraSite Control

After you create the XQuery search module, you must upload the new XQuery module to the CentraSite repository.

➤ To upload the XQuery module

1. In CentraSite Control, go to **Asset Catalog > Search**.

This opens the Asset Search page.

2. Click the **XQuery** tab.
3. Type the new XQuery code in the text box.
4. Click **Save**.

Uploading XQuery Module through Command Line

To upload a custom XQuery module through the CentraSite Command Line (.cmd) Interface, you must have the CentraSite Administrator role.

CentraSite provides a Java tool named `LoadModule.jar` for this purpose.

- Run the Java tool `LoadModule.jar`.

The syntax is of the format: `C:\SoftwareAG\CentraSite\utilities>CentraSiteToolbox.cmd LoadModule.jar <CentraSite URL> <admin user id> <password> <module file name> [<module name>]`

The input parameters are:

Parameter	Description
CentraSite URL	The URL of the CentraSite registry. For example, <code>http://localhost:53307/CentraSite/CentraSite</code> .
admin user id	The user ID of a registered CentraSite user who has the CentraSite Administrator role. For example, <code>Administrator</code> .
password	The password for the CentraSite user identified by the parameter <code>admin user id</code> .
module file name	Name of the XQuery file that contains the module and functions to upload. If the file name contains white spaces, enclose the name with " ".
module name	Name of the XQuery module to upload. If the module name contains white spaces, enclose the name with " ".

Example (all in one line):

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteToolbox.cmd LoadModule.jar
http://localhost:53307/CentraSite/CentraSite DOMAIN\admin pAsSw0rD MyReport.xquery
```

Writing Saved Search XML

This section explains how you write a saved search XML file that makes use of the above XQuery function and defines the required search parameters and result attributes.

Saved Search XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<Search type="report">
  <Expression name="getAPIsWithAPIKeys" namespace="http://namespaces.CentraSite.com/m">
    <Params>
      <Param type="xs:integer" isMultiValued="false" displayName="INMCL_PARAM_LIMIT_C">
      </Param>
    </Params>
  </Expression>
  <ResultAttributes>
    <Attribute type="xs:string">
      <Name>assetName</Name>
      <DisplayName>INMCL_COLUMN_ASSET_NAME</DisplayName>
      <Link type="AssetDetail">
        <Param>assetKey</Param>
      </Link>
    </Attribute>
    <Attribute type="xs:string">
      <Name>assetDesc</Name>
      <DisplayName>INMCL_COLUMN_ASSET_DESC</DisplayName>
    </Attribute>
    <Attribute type="xs:integer">
      <Name>apiKeysCountForAsset</Name>
      <DisplayName>No of API Keys</DisplayName>
    </Attribute>
  </ResultAttributes>
</Search>
```

Note:

Make sure that the value of XQuery function name and module namespace in the saved search XML file exactly match with the values specified in the XQuery module definition. Let's call this saved search XML file as TopXAssetsBasedonAPIKeys.xml.

Uploading Custom Reporting Search through Command Line

Pre-requisites:

To upload a custom reporting search through the CentraSite Command Line Interface, you must have the CentraSite Administrator role.

After you create the XQuery search module and search XML file, you must upload them to the CentraSite repository.

CentraSite provides a command tool named `add Search` for this purpose.

Important: CentraSite relies on file extensions to determine a file's type. When you upload a saved search XML file or an XQuery module from your local machine using the command line, be sure you specify the name of the file along with its extension (.xml or .xquery) so that CentraSite can determine the file's type and mark it correctly in the repository.

➤ To upload a custom reporting search

- Run the command `add Search`.

The syntax is of the format: `C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd add search [-url <CENTRASITE-URL>] -user <USER-ID> -password <PASSWORD> [-module <XQUERY-MODULE>] [-overwrite <CONFIRM-OVERWRITE>] [-scope <SCOPE>] -savedSearchFile <SAVED-SEARCH-FILE>`

The input parameters are:

Parameter	Description
CENTRASITE-URL	(Optional). The URL of the CentraSite registry. For example, <code>http://localhost:53307/CentraSite/CentraSite</code> .
USER-ID	The user ID of a registered CentraSite user who has the CentraSite Administrator role. For example, <code>Administrator</code> .
PASSWORD	The password for the registered CentraSite user identified by the parameter <code>USER-ID</code> .
XQUERY-MODULE	The absolute or relative path to the XQuery module.
CONFIRM-OVERWRITE	Specifies whether to overwrite an existing report XQuery module with the new XQuery module. The possible values: <ul style="list-style-type: none"> ■ <code>true</code> - Overwrite existing XQuery module. ■ <code>false</code> - Do not overwrite existing XQuery module.
SCOPE	The folder to store the newly added saved search in CentraSite.

Parameter	Description
	<p>The possible values are:</p> <ul style="list-style-type: none"> ■ <code>global</code> - This option stores the search information in "Default Organization" folder. ■ <code>org</code> - This option stores the search information in the user's organization folder. ■ <code>user</code> (default value) - This option stores the search information in the user folder.
<code>SAVED-SEARCH-FILE</code>	Name of the XML file which contains the saved search.

Example (all in one line):

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd add Search -url
http://localhost:53307/CentraSite/CentraSite -user Administrator -password manage
-savedSearchFile MySearch.xml -module MyPortlet.xquery -overwrite true -scope org
```

The response to this command could be:

```
Executing the command : add Search
Successfully executed the command : add Search
```

After the command executes successfully, the newly created portlet, MyPortlet, is displayed in the **Data Feed** list in **Add New Portlet** dialog box.

Activating Custom Reporting Search through CentraSite Business UI

After you upload the custom reporting search to CentraSite repository, you must activate this reporting search through CentraSite Business UI.

➤ To activate the custom reporting search

1. In CentraSite Business UI, click the **Welcome** link in the upper-right corner of the header area.

The Welcome page displays a list of portlets that are configured for your view.

2. Click the **Configure** link (below the label **Welcome to CentraSite Business UI**).

The **Configure your Welcome Page** dialog box opens to display the list of portlets that are available to you.

3. In the **Configure Your Welcome Page** dialog box, click the **Add a portlet** link.

This opens the **Add a portlet** dialog box.

4. In the **Add Portlet** dialog box, provide the required information for each of the displayed data fields. For more information about specifying the data fields, see the *CentraSite User's Guide*.
5. In the **Data Feed** box, select the custom reporting search that you uploaded to the CentraSite repository.
6. Click **OK**.

The portlet that you just created with the custom reporting search is added to the CentraSite Registry Repository, and you are redirected to the **Configure Your Welcome Page** dialog box.

By default, this newly created portlet is disabled and is not displayed in the Welcome page.

7. In the **Configure Your Welcome Page** dialog box, select the newly created portlet to add to your Welcome page.
8. Click **OK**.

The newly created portlet is displayed in your personalized Welcome page.

Deleting Custom Reporting Search through Command Line

Pre-requisites:

To delete a custom reporting search through the CentraSite Command Line Interface, you must have the CentraSite Administrator role.

CentraSite provides a command tool named `delete Search` for this purpose.

➤ To delete a custom reporting search

- Run the command `delete Search`.

The syntax is of the format: `C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd delete search [-url <CENTRASITE-URL>] -user <USER-ID> -password <PASSWORD> [-savedSearch <SAVED-SEARCH>] [-scope <SCOPE>]`

The input parameters are:

Parameter	Description
CENTRASITE-URL	(Optional). The URL of the CentraSite registry. For example, <code>http://localhost:53307/CentraSite/CentraSite</code> .
USER-ID	The user ID of a registered CentraSite user who has the CentraSite Administrator role. For example, Administrator.

Parameter	Description
PASSWORD	The password for the registered CentraSite user identified by the parameter USER-ID.
SAVED-SEARCH	Name of the saved search you want to delete.
SCOPE	<p>Name of the folder that contains the saved search you want to delete.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> ■ global - List of saved searches that belong to the Default Organization. ■ org - List of saved searches that belong to the logged-in user's organization. ■ user - Default. List of saved searches that belong to the logged-in user.

Example (all in one line):

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd delete Search -url
http://localhost:53307/CentraSite/CentraSite -user Administrator -password manage
-savedSearch "My Saved Search" -scope org
```

The response to this command could be:

```
Executing the command : delete Search
Successfully executed the command : delete Search
```

Displaying List of Reporting Searches

Pre-requisites:

To display the list of all reporting searches through the CentraSite Command Line Interface, you must have the CentraSite Administrator role.

CentraSite provides a command tool named `list Search` for this purpose.

➤ To display the list of reporting searches

- Run the command `list Search`.

The syntax is of the format: `C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd list search [-url <CENTRASITE-URL>] -user <USER-ID> -password <PASSWORD> [-scope <SCOPE>]`

The input parameters are:

Parameter	Description
CENTRASITE-URL	(Optional). The URL of the CentraSite registry. Default value is <code>http://localhost:53307</code> .
USER-ID	The user ID of a registered CentraSite user who has the CentraSite Administrator role. For example, Administrator.
PASSWORD	The password for the registered CentraSite user identified by the parameter USER-ID.
SCOPE	<p>The folder to store the newly added saved search in CentraSite.</p> <p>The possible values are:</p> <ul style="list-style-type: none">■ <code>global</code> - This option stores the search information in "Default Organization" folder.■ <code>org</code> - This option stores the search information in the user's organization folder.■ <code>user</code> (default value) - This option stores the search information in the user folder.

Example (all in one line):

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd list Search -url
http://localhost:53307/CentraSite/CentraSite -user Administrator -password manage
-scope org
```

Obtaining Details of Reporting Search

Pre-requisites:

To obtain the details of a reporting search through the CentraSite Command Line Interface, you must have the CentraSite Administrator role.

CentraSite provides a command tool named `get Search` for this purpose.

> To display the details of a reporting search

- Run the command `get Search`.

The syntax is of the format: `C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd get search [-url <CENTRASITE-URL>] -user <USER-ID> -password <PASSWORD> -savedSearch <SAVED-SEARCH> [-scope <SCOPE>]`

The input parameters are:

Parameter	Description
CENTRASITE-URL	(Optional). The URL of the CentraSite registry. Default value is <code>http://localhost:53307</code> .
USER-ID	The user ID of a registered CentraSite user who has the CentraSite Administrator role. For example, Administrator.
PASSWORD	The password for the registered CentraSite user identified by the parameter USER-ID.
SAVED-SEARCH	Name of the saved search in CentraSite.
SCOPE	<p>The folder to store the newly added saved search in CentraSite.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> ■ <code>global</code> - This option stores the search information in "Default Organization" folder. ■ <code>org</code> - This option stores the search information in the user's organization folder. ■ <code>user</code> (default value) - This option stores the search information in the user folder.

Example (all in one line):

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd get Search -url
http://localhost:53307/CentraSite/CentraSite -user Administrator -password manage
-savedSearch MySearch -scope org
```

Copying Reporting Search

Pre-requisites:

To create a copy of an existing reporting search through the CentraSite Command Line Interface, you must have the CentraSite Administrator role.

A saved search can become quite complex, especially if it contains several attributes. Instead of creating a new saved search "from scratch", it is sometimes easier to copy an existing search that is similar to the one you need and edit the copy.

CentraSite provides a command tool named `copy Search` for this purpose.

➤ To copy a custom reporting search

- Run the command `copy Search`.

The syntax is of the format: `C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd copy search [-url <CENTRASITE-URL>] -user <USER-ID> -password <PASSWORD> -savedSearch`

```
<SAVED-SEARCH> [-overwrite <CONFIRM-OVERWRITE>] [-scope <SCOPE>] [-targetuser  
<TARGET-USER>]
```

The input parameters are:

Parameter	Description
CENTRASITE-URL	(Optional). The URL of the CentraSite registry. Default value is <code>http://localhost:53307</code> .
USER-ID	The user ID of a registered CentraSite user who has the CentraSite Administrator role. For example, Administrator.
PASSWORD	The password for the registered CentraSite user identified by the parameter USER-ID.
SAVED-SEARCH	Name of the saved search in CentraSite.
SCOPE	<p>The folder to store the newly added saved search in CentraSite.</p> <p>The possible values are:</p> <ul style="list-style-type: none">■ <code>global</code> - This option stores the search information in "Default Organization" folder.■ <code>org</code> - This option stores the search information in the user's organization folder.■ <code>user</code> (default value) - This option stores the search information in the user folder.
TARGET-USER	The name or UUID of a CentraSite user to whom you want to transfer ownership of the objects that are referenced by the organization identified by the parameter <code>SAVED-SEARCH</code> .

Example (all in one line):

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd copy Search -url  
http://localhost:53307/CentraSite/CentraSite -user Administrator -password manage  
-savedSearch MySearch -scope org -targetuser SAGDomain\Claire
```

Sample XQueries for Reporting Searches

CentraSite provides a collection of sample queries which can be used to fetch the search results directly or in a modified version against the CentraSite server. For the purpose of demonstration, let us suppose that our search data is based on an online fictional store selling a wide range of sheet music. In order to make this music store available online, CentraSite provides a set of suitable services. The services operate on a common database that includes the required information about the music sheets offered, the customers, and the orders. The services and their usage are exposed to the potential callers.

XQueries for Retrieving Primary Objects

Primary objects include services, organizations, classificationSchemes, concepts, packages, externalLinks, and instances of user-defined object-types. There are a number of primary object types in the JAXR data model representation of CentraSite database. Also, instances of the user-defined object types become primary objects, since when you add a new type to CentraSite, a respective schema is created that defines the new type as a new doctype to the CentraSite database. Instances of the primary object types are represented as XML documents in the CentraSite database collection.

The sample XQuery to find all organizations in CentraSite, for example, is:

```
for $organization in collection("CentraSite")/*:organization
return $organization
```

The document element name pattern `*:organization` finds all documents with a local-name `organization` in any namespace. A proper namespace allows the XQuery processor more means of optimization (index utilization, for example), and therefore we enhance the XQuery definition as:

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr"
for $organization in collection("CentraSite")/jaxr:organization
return $organization
```

To find a specific organization, we include a where clause in the above XQuery definition. Note that most of the elements to be retrieved (as the key element holding the unique UDDI key) are also in the JAXR namespace.

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
for $organization in collection("CentraSite")/jaxr:organization
where $organization/jaxr:key = "uddi:4299de0d-51a4-11de-9115-99700cdcef42"
return $organization
```

To return a more sophisticated result than just the retrieved element as it is contained in the CentraSite database, define a result structure as the XQuery result and insert the data taken from the element. The next XQuery returns a generated element `organization` containing the organization's name and its owner. The owner is identified as a user (another primary object type) and is defined by the organization's sub-element `owner`.

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
for $organization in collection("CentraSite")/jaxr:organization
where $organization/jaxr:key = "uddi:4299de0d-51a4-11de-9115-99700cdcef42"
return
  <organization>
    <name> { string(($organization/jaxr:name/jaxr:localString)[1]) } </name>
    <owner> {
      let $owner := collection("CentraSite")/*:user
        [jaxr:key = $organization/jaxr:owner]
      return string(($owner/jaxr:name/jaxr:localString)[1])
    } </owner>
  </organization>
```

This results in a newly generated element node containing elements for the organization's name and owner as below:

```
<xq:result xmlns:xq="http://namespaces.softwareag.com/tamino/XQuery/result">
  <organization>
    <name>Mozart Music GmbH</name>
    <owner>INTERNAL\Administrator</owner>
  </organization>
</xq:result>
```

To find a primary object's name, we declare and use a user-defined XQuery function:

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
declare function local:getName($node as node()) as xs:string {
  string(($node/jaxr:name/jaxr:localString)[1])
};
for $organization in collection("CentraSite")/jaxr:organization
where $organization/jaxr:key = "uddi:4299de0d-51a4-11de-9115-99700cdcef42"
return
  <organization>
    <name> { local:getName($organization) } </name>
    <owner> {
      let $owner := collection("CentraSite")/*:user
      [jaxr:key = $organization/jaxr:owner]
      return local:getName($owner)
    } </owner>
  </organization>
```

XQueries for Retrieving Embedded Objects

Embedded objects (for example, slots, classifications, and associations) occur embedded within the primary objects.

As an example, let us look at the slots provided with the service `MusicQuoteService`. The `MusicQuoteService` provides information on the music sheets by a composer. This service uses the composer's name as an input and returns the list of music sheets by that specific composer. A sheet, in this case, is a paired value consisting of a sheet's title and an ID that identifies the sheet in the online store database.

Let us assume that we bundle this service into different colour schemes and attach the property colour as a slot (key-value pair) to the service.

The screenshot shows the CentraSite Community Edition web interface. The top navigation bar includes links for Home, Asset Catalog, Policies, Administration, and Reports. The main content area displays the configuration for the MusicQuoteService. It includes fields for Name (MusicQuoteService), Description, Version (1.0), and metadata such as Created (2011-11-08 02:19 PM), Last Modified (2011-11-08 02:28 PM), Organization (Mozart Music GmbH), and Owner (INTERNAL\Administrator). Below this, there is a table with columns for Name, Namespace, and Value. The table contains one entry: 'colour' with a value of 'blue'. The interface also features tabs for Summary, Technical Details, Specification, Support, Consumers, Permissions, Classifications, and Associations.

You may use an XQuery function to find the colour property:

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
declare function local:getName($node as node()) as xs:string {
  string($node/jaxr:name/jaxr:localString)
};
for $service in collection("CentraSite")/jaxr:service
where local:getName($service) = "MusicQuoteService"
return $service/jaxr:instanceSlots/colour
```

This XQuery function returns :

```
<colour xmlns="">blue</colour>
```

When dealing with function calls, the optimizer sometimes decides to inline the XQuery function. That is, before an XQuery gets executed, a function call is replaced with the body of the respective function. In this XQuery, the function call to `local:getName` might not get inline during optimization. Optionally, you can precede the function declaration with an inline XQuery pragma.

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
{?inline?}
declare function local:getName($node as node()) as xs:string {
  string($node/jaxr:name/jaxr:localString)
};
for $service in collection("CentraSite")/jaxr:service
where local:getName($service) = "MusicQuoteService"
return $service/jaxr:instanceSlots/colour
```

The optimization pragma `{?optimization inline="full"?}` at the beginning of the XQuery would make all user-defined functions to be inline except those that directly or indirectly reference themselves.

Some of the embedded object types are related to specific primary object types as `serviceBindings` that occur as sub elements of the services. An XQuery using the `local-name` function would retrieve the sub elements that exist under the current element in an XML document:

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
declare function local:getName($node as node()) as xs:string {
  string($node/jaxr:name/jaxr:localString)
};
for $service in collection("CentraSite")/jaxr:service
where local:getName($service) = "MusicQuoteService"
return for $subelement in $service/*
  return local-name($subelement)
```

This XQuery function yields:

```
<xq:result xmlns:xq="http://namespaces.softwareag.com/tamino/XQuery/result">
  <xq:value>key</xq:value>
  <xq:value>owner</xq:value>
  <xq:value>name</xq:value>
  <xq:value>description</xq:value>
  <xq:value>submittingOrganization</xq:value>
  <xq:value>externalLinks</xq:value>
  <xq:value>classifications</xq:value>
  <xq:value>majorVersion</xq:value>
  <xq:value>minorVersion</xq:value>
  <xq:value>stability</xq:value>
  <xq:value>status</xq:value>
  <xq:value>providingOrganization</xq:value>
  <xq:value>serviceBindings</xq:value>
  <xq:value>instanceSlots</xq:value>
</xq:result>
```

XQueries for Retrieving Classifications

The purpose of classifying objects is to allow a means of semantic retrieval. Let us assume that we have the following taxonomy classifications to describe the different types of music:

- e: Classical Music
- ec: Classical Chamber Music
- ecb: Brass Chamber Music
- eo: Classical Music for Orchestras
- u: Non-Classical Music
- us: Music for Small Bands
- ub: BigBand Music

This enables the customer to retrieve specific music sheets as classified in the online store. An XQuery function to retrieve the concept object representing the taxonomy classification ecb (Brass Chamber Music):

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
for $concept in collection("CentraSite")/jaxr:concept
where $concept/jaxr:name/jaxr:localString = "ecb"
return $concept
```

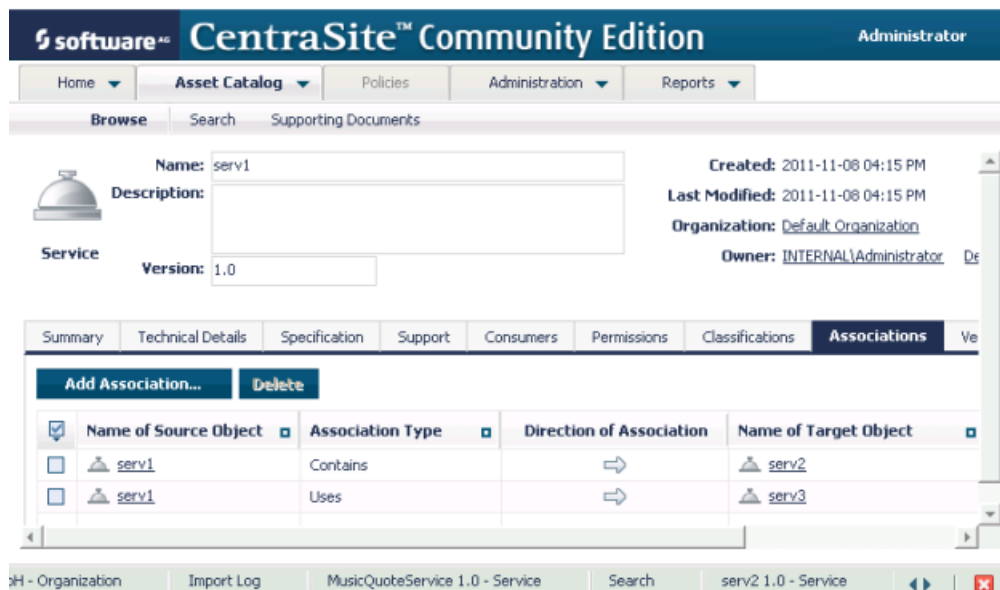
Make sure that you do not use the user-defined function `local:getName` in the above XQuery function, because this function assumes that every such concept only has one name which cannot be expected in an internationalized environment.

The following XQuery function retrieves the services that are classified by this concept to search for music sheets applicable to Brass Chamber Music.

```
declare namespace jaxr = "http://namespaces.CentraSite.com/Schema/jaxr";
declare function local:getName($node as node()) as xs:string {
  string($node/jaxr:name/jaxr:localString)
};
let $concept := collection("CentraSite")/jaxr:concept[jaxr:name/
                                                         jaxr:localString="ecb"]
for $service in collection("CentraSite")/jaxr:service
where $service//jaxr:classification/jaxr:concept = $concept/jaxr:key
return local:getName($service)
```

XQueries for Retrieving Associations

Let us assume a service `serv1` relates to two other services `serv2` and `serv3` using the predefined association types `Contains` and `Uses` respectively.



The following XQuery function finds all registry entries using the `Contains` association from the service `serv1`. Initially, XQuery finds the concept corresponding to association type `Contains` and the service `serv1`. Associations are stored with the primary objects they start from, that is, the two associations starting from `serv1` are stored as `cs:association` descendants of `serv1`.

```
declare namespace cs="http://namespaces.CentraSite.com/Schema/jaxr"
let $concept := collection("CentraSite")/cs:concept
               [cs:name/cs:localString = "Contains"]/cs:key
let $service := collection("CentraSite")/
               cs:service[cs:name/cs:localString="serv1"]
for $target in $service//cs:association[cs:associationType=$concept]/
               cs:targetObject
return collection("CentraSite")/cs:*[cs:key=$target]
```

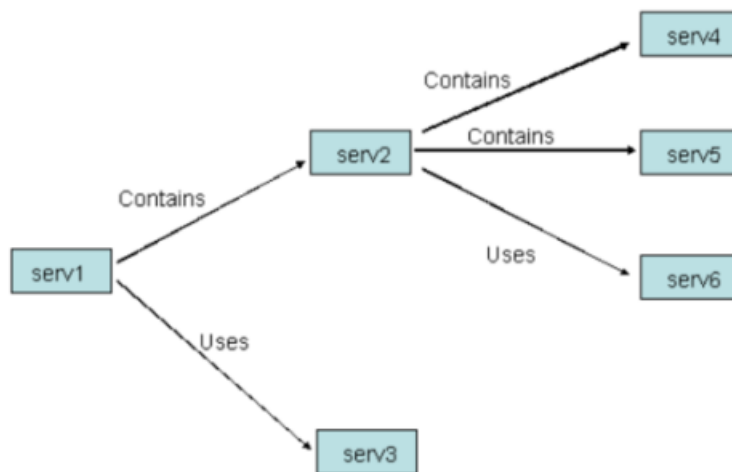
This XQuery function returns the complete service serv2. The next XQuery function uses the local function `getName` to only return the services name.

```
declare namespace cs="http://namespaces.CentraSite.com/Schema/jaxr"
declare function local:getName($node as node()) as xs:string
{
  xs:string($node/cs:name/cs:localString)
}
let $concept := collection("CentraSite")/cs:concept
               [cs:name/cs:localString = "Contains"]/cs:key
let $service := collection("CentraSite")/cs:service
               [cs:name/cs:localString="serv1"]
for $target in $service//cs:association
               [cs:associationType=$concept]/cs:targetObject
return local:getName(collection("CentraSite")/cs:*[cs:key=$target])
```

The following XQuery function uses another local function `followAssocOnce` to follow the association.

```
declare namespace cs="http://namespaces.CentraSite.com/Schema/jaxr"
declare function local:getName($node as node()) as xs:string
{
  xs:string($node/cs:name/cs:localString)
}
declare function local:followAssocOnce($entry as node(),
                                       $assoc as xs:string)
  as node()*
{
  for $key in $entry//cs:association
               [cs:associationType=$assoc]/cs:targetObject
  return collection("CentraSite")/*[cs:key=$key]
}
let $concept := collection("CentraSite")/cs:concept
               [cs:name/cs:localString = "Contains"]/cs:key
let $service := collection("CentraSite")/cs:service
               [cs:name/cs:localString="serv1"]
for $target in local:followAssocOnce($service,$concept)
return local:getName($target)
```

Let us further assume that `serv2` points to three other entries, namely the services `serv4` and `serv5` with `Contains` association and the service `serv6` with `Uses` association.



The following XQuery function includes a recursive local function `followAssocMultiple`. This function retrieves all entries that are reachable by following a specific association from a specific entry. When this XQuery function is applied upon the service `serv1` and association `Contains`, it returns the services `serv2`, `serv4`, and `serv5`.

This XQuery function, if applied on an entry that is part of or forms a closed loop system, yields a stack overflow.

```

declare namespace cs="http://namespaces.CentraSite.com/Schema/jaxr";
declare function local:getName($node as node()) as xs:string
{
  xs:string($node/cs:name/cs:localString)
}
declare function local:followAssocOnce($entry as node(),
                                       $assoc as xs:string)
  as node()*
{
  for $key in $entry//cs:association
    [cs:associationType=$assoc]/cs:targetObject
  return collection("CentraSite")/*[cs:key=$key]
}
declare function local:followAssocMultiple($entry as node(),
                                           $assoc as xs:string)
  as node()*
{
  for $target in local:followAssocOnce($entry,$assoc)
  return ($target,local:followAssocMultiple($target,$assoc))
}
let $concept := collection("CentraSite")/cs:concept
               [cs:name/cs:localString = "Contains"]/cs:key
let $service := collection("CentraSite")/cs:service
               [cs:name/cs:localString="serv1"]
for $target in local:followAssocMultiple($service,$concept)
return local:getName($target)
  
```

The following XQuery function is applicable on an entry that is part of or forms a closed loop system. This requires that the recursive function takes three parameters. The first parameter is the list of already found nodes. The second parameter holds the current nodes, that is, the nodes found

in the previous step. Each time you call the XQuery function, it finds the nodes reached directly from the current nodes. These nodes become the new nodes unless they have previously been found.

```
declare namespace cs="http://namespaces.CentraSite.com/Schema/jaxr";
declare function local:getName($node as node()) as xs:string
{
  xs:string($node/cs:name/cs:localString)
}
declare function local:followAssocOnce($entry as node(),
                                       $assoc as xs:string)
  as node()*
{
  for $key in $entry//cs:association
    [cs:associationType=$assoc]/cs:targetObject
  return collection("CentraSite")/*[cs:key=$key]
}
declare function local:followAssocMultiple($current as node()*,
                                           $assoc as xs:string)
  as node()* {
  let $first := for $node in $current
    return local:followAssocOnce($node,$assoc)
  return local:followAssocMultipleRecursive($first,$assoc)
}
declare function local:followAssocMultipleRecursive(
  $old as node()*,
  $current as node()*,
  $assoc as xs:string) as node()* {
  if ($current)
  then let $new := for $entry in $current
    return local:followAssocOnce($entry,$assoc)
    let $nextold := $old union $current
    let $nextcurrent := $new except $nextold
    return local:followAssocMultipleRecursive($nextold,
                                              $nextcurrent,
                                              $assoc)
  else $old
}
let $concept := collection("CentraSite")/cs:concept
  [cs:name/cs:localString = "Contains"]/cs:key
let $service := collection("CentraSite")/cs:service
  [cs:name/cs:localString="serv1"]
for $target in local:followAssocMultiple($service,$concept)
return local:getName($target)
```

When following relations in backward direction, retrieval of entries is slightly more complex. To retrieve all entries from which a relation \$assoc traverses to a specific entry \$entry, find all the entries and check its relations. The following XQuery function uses a local function followAssocOnceBackwards to find all entries starting from serv4, traversing the association Contains in backward direction, and yields serv2.

```
declare namespace cs="http://namespaces.CentraSite.com/Schema/jaxr"
declare function local:getName($node as node()) as xs:string
{
  xs:string($node/cs:name/cs:localString)
}
declare function local:followAssocOnceBackwards($entry as node(),
                                              $assoc as xs:string)
```

```
as node()*  
{  
  for $everyentry in collection("CentraSite")/*  
  for $target in $everyentry//cs:association  
    [cs:associationType=$assoc]/cs:targetObject  
  where $target = $entry/cs:key  
  return $everyentry  
}  
let $concept := collection("CentraSite")/cs:concept  
  [cs:name/cs:localString = "Contains"]/cs:key  
let $service := collection("CentraSite")/cs:service  
  [cs:name/cs:localString="serv4"]  
for $target in local:followAssocOnceBackwards($service,$concept)  
return local:getName($target)
```


7 Application Framework

■ Introduction to Application Framework	234
■ Configuring Application Framework	236
■ Mapping Beans to Registry Objects with Annotations	238
■ Querying the Registry	247
■ Event Mechanism	256
■ Asset Types	257
■ Association Types	258
■ Lifecycle Management	259
■ Revision Management	261
■ Multi-User Scenarios	262
■ Setting the Classpath	263
■ Examples	263

Introduction to Application Framework

The CentraSite Application Framework (CSAF) provides a programming model for developing custom extensions on top of CentraSite. It supports JAXR (Java API for XML Registries) and extends the CentraSite JAXR-based API and the Pluggable UI - the framework on which the CentraSite Control UI is built.

It contains two independent parts: the persistence framework and the validation framework.

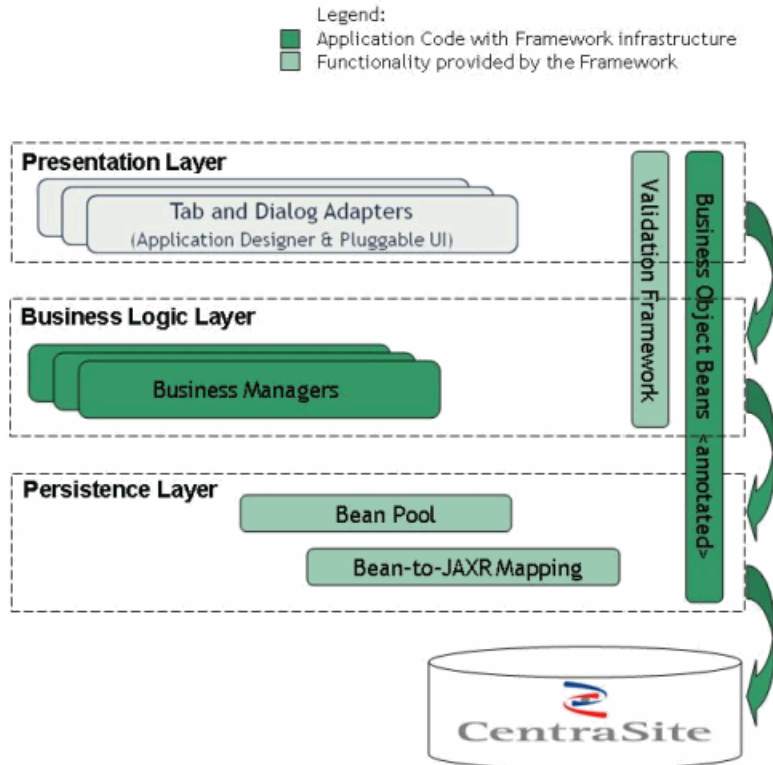
The persistence framework provides the ability to operate on registry data using JavaBeans instead of the JAXR-based API. This is done in a fashion similar to object-relational mapping tools such as Hibernate or Java Persistence API. In this case, Java Beans are mapped to registry objects. All this is done declaratively using Java5 Annotations.

This framework was created with the intention of making it easier to work with registries that support the JAXR-based interface, such as CentraSite. Its usage does not require in general any specific or deep knowledge of this API.

A direct benefit of this is shortened application development time.

The validation framework provides an extensible mechanism for validating Java beans. Multiple numbers of constraints can be attached to each bean. The notion of scopes is also supported, that is, constraints apply only when specific conditions about the bean are met.

The architecture of a common CentraSite application extension developed using CSAF:



There are two major points that have to be clear in order to understand how the persistence framework works, namely how the bean model is built based on the RegistryBean interface and the BeanPool.

RegistryBean

The RegistryBean (`com.softwareag.centrasite.appl.framework.beans.RegistryBean`) interface has to stay on top of each bean model hierarchy.

It contains the properties that a registry object would have, namely a key and a name. Implementing is the only restriction the framework on the application bean model. The user can use DynamicRegistryBean (`com.softwareag.centrasite.appl.framework.beans.DynamicRegistryBean`) for implementation of RegistryBeans.

It implements RegistryBean and RevisionBean (`com.softwareag.centrasite.appl.framework.beans.RevisionBean`), which is the revision-aware extension of the RegistryBean interface.

There is one more option here. If the registry bean needs to be lifecycle-aware, then the user should use the `com.softwareag.centrasite.appl.framework.lcm.beans.LifecycleAware` interface instead of RegistryBean.

Its implementation is handled by `com.softwareag.centrasite.appl.framework.lcm.beans.LCAwareDynamicRegistryBean`.

BeanPool

The BeanPool (`com.softwareag.centrasite.appl.framework.persistence.BeanPool`) is the main interface with which the application interacts in order to use the persistence framework.

All CRUD (create, read, update, delete) operations search via this interface and registry queries are done via this interface. The user must be aware that the BeanPool instances are not thread safe. There can be only one beanPool per SessionContext. CSAF provides the functionality to create beanPool instances by using `SessionContext.createBeanPool()`. The beanPool can be accessed by `SessionContext.getCurrentBeanPool()`. This method returns the BeanPool instance that is associated with the given context. The CurrentBeanPoolContext interface defines the contract for implementations which knows how to scope the notion of a current bean pool. An implementation of this interface is provided as `ThreadLocalCurrentBeanPoolContext`, which maintains current bean pools for the given execution thread. This functionality is extensible, so users can create their own context by implementing `CurrentBeanPoolContext`.

StandaloneRegistryProvider

In order to obtain a connection to the repository, an instance of StandAloneRegistryProvider must be created. This registry provider has several important parameters for its creation that will affect the functionality of CSAF. CSAF supports several constructors which exclude some of the properties and use their default values instead. The constructor with full parameter list is:

```
StandaloneRegistryProvider(String registryUrl, String user,
    String password, boolean browserBehaviour){}
```

registryUrl	The URL of the CentraSite registry. For example, <code>http://localhost:53307/CentraSite/CentraSite</code> .
user	The user ID of a registered CentraSite user who has the CentraSite Administrator role. For example, <code>Administrator</code> .
password	The password of the user identified by the parameter <code>-user</code> .
browserBehaviour	Sets the <code>com.centrasite.jaxr.BrowserBehaviour</code> property of the connection factory. To enable type management, this flag must be set to <code>true</code> ; to enable RevisionManagement it must be <code>false</code> . Default value is <code>false</code> .

Example for creating a BeanPool instance by using SessionContext and StandaloneRegistryProvider

```
SessionContext context = null;
RegistryProvider provider = null; try {
    provider = new StandaloneRegistryProvider(registryUsername,
                                             registryPassword, true);

    Configuration conf = new Configuration();
    conf.setRegistryProvider(provider);
    conf.addBeanType(Item.class);
    conf.addBeanType(Action.class);
    conf.addBeanType(Entry.class);
    conf.addBeanType(ExternalLink.class);
    context = SessionContext.createInstance(conf);
} catch (CSAppFrameworkException e) {
    // Do something with the exception
}

BeanPool beanPool = context.getCurrentBeanPool();
```

Configuring Application Framework

You can configure the CentraSite Application Framework via the Configuration object (`com.softwareag.centrasite.appl.framework.Configuration`).

The following can be configured here:

- Bean types managed by CSAF
- Persistence mode
- Bean mode
- Maximum concept cache size
- Cache scope
- Re-reading of outdated objects

Additionally, the configuration object supports a generic property: key/name pair. It can be used to configure any of the above mentioned properties generically.

After the Configuration object has been initialized, it can be passed to the `com.softwareag.centrasite.appl.framework.SessionContext.createInstance()` method, which creates a SessionContext instance.

This instance can then create `com.softwareag.centrasite.appl.framework.persistence.BeanPool` instances and can be used for the lifetime of the application.

Bean Types Managed by CSAF

The framework keeps an internal data model for user-defined bean classes, for example, bean classes that extend the `com.softwareag.centrasite.appl.framework.beans.RegistryBean` interface.

After the bean interfaces have been defined as Java classes having the `@RegistryObject` annotation, they must be registered by calling the `Configuration.addBeanType(java.util.Class)` method.

In principle, calling `Configuration.addBeanType(java.util.Class)` for each bean class is not mandatory, since CSAF tries to process this information (configuration) at runtime when required. Nevertheless, it is still highly recommended because there are cases in which it is not possible to obtain the mapping information at runtime, for example, when performing a search in the registry.

Bean Modes

The framework supports two bean modes: BACKED and SIMPLE (`com.softwareag.centrasite.appl.framework.persistence.BeanMode`). This mode specifies how the beans interact with the underlying implementation of the API supporting JAXR.

When using the SIMPLE mode, data from the bean is transferred to the registry object only when the user explicitly requests this by calling one of the BeanPool methods (`update()`, `flush()`, `delete()`).

When using the BACKED mode, data from the bean is transferred to the registry object immediately after it is set in the bean. The advantage of this is that extra features such as locking and caching can be used.

Note:

SIMPLE mode is deprecated; BACKED mode should always be preferred.

Persistence Modes

The framework supports two persistence modes: FULL and MAP_ONLY. This mode specifies how and whether the data will be persisted in the registry.

When using the FULL mode, the data is entirely persisted in the registry. This is the default mode.

When using the MAP_ONLY mode, the data is not persisted in the registry at all; it is just mapped from the bean to the registry object. It is assumed that the persistence is done outside of the framework. This is used, for example, in a Pluggable UI environment, and applies to any custom extension of the CentraSite UI. In this, the Pluggable UI takes care of storing the object in the registry.

Cache Configuration

Two properties of the caching can be configured: the maximum concept cache size and the cache scope. Both parameters configure the concept mapping cache within the framework.

The default value for the cache size is 1000.

There are two available scopes for the cache:

- APPLICATION: There is one cache for the whole application.
- SESSION: Each session has its own cache.

Re-reading Outdated Objects

The framework provides support to re-read outdated registry beans automatically. This is controlled by the `Configuration.PROP_AUTO_REREAD_OUTDATED_OBJECTS` property. Possible values are `true` and `false`.

If the property value is set to `true` then when an outdated object is modified by the system it is automatically re-read from the registry, that is, reverted to the latest state in the database, before applying any changes. Otherwise the user receives the following exception when performing the modification:

```
com.softwareag.centrasite.appl.framework.persistence.ObjectOutdatedException
```

Note:

If this feature is turned on the current client of CSAF overrides any changes made by another client.

Mapping Beans to Registry Objects with Annotations

The beans are mapped to registry objects using Java5 Annotations.

Each bean from the application bean model has to extend or implement the `RegistryBean` (`com.softwareag.centrasite.appl.framework.beans.RegistryBean`) interface. If an interface extends the `RegistryBean` interface, an implementation must be provided and specified using the `@Bean` annotation:

```
@RegistryObject(objectTypeName="{http://namespaces.CentraSite.com/csaf}Item")
@Bean(implementationClass = "...")
public interface Item extends RegistryBean{
    ...
}
```

The following table describes the annotations currently supported by the CentraSite Application framework:

Annotations and Description	Scope	Properties
<code>@RegistryObject</code> Maps a bean to a registry object with a specific object type.	Type	<code>objectTypeName</code> (optional) – the name of the object type of the registry object. <code>objectTypeKey</code> (optional) – the key of the object type. At least one of the properties must be specified.

Annotations and Description	Scope	Properties
<p>@Property</p> <p>Maps a bean property to a registry object property. The properties should have the same type. The mapper does not provide type conversion, except for JAXR InternationalString to/from String.</p>	Method	<p>target (optional) – the name of the target property in the registry object. The property must be a standard property of a predefined JAXR-based object type. If the target property is not specified, it is assumed that it matches the name of the bean property.</p>
<p>@Slot</p> <p>Maps a bean property to a registry object slot. Multivalue slots are supported. Also provided are type conversion slot values which are string to integer, Boolean, date, timestamp, and Calendar.</p>	Method	<p>Name (mandatory) – the name of the slot to which this property is to be mapped. The JAXR-based property being mapped can be custom defined, or the JAXR-based object type that this property comes from can be custom.</p> <p>targetType (optional) – specifies the type of the bean property. It is used when the property is a collection and thus the mapping cannot guess the underlying property.</p> <p>type (optional) – the type of the bean property. Supported types are BOOLEAN, DATE, CALENDAR, TIMESTAMP, INTEGER, LONG, and AUTO. The latter allows the mapper to guess the property type.</p>
<p>@Slots</p> <p>Maps all slots of a registry object to a bean property (Collection).</p>	Method	<p>targetType (mandatory) - the type of bean that is to be mapped to a single slot.</p>
<p>@SlotProperty</p> <p>Used in conjunction with the @Slots property. Maps the properties of the bean of the type specified as target type with the @Slots annotation. A slot has a name, slot type and values. All these properties can be mapped using this annotation.</p>	Method	<p>target (mandatory) – can be one value from the enum SlotPropertyName – NAME, SLOT_TYPE, VALUES.</p>
<p>@TelephoneNumbers</p> <p>Maps a bean property to the TelephoneNumber object from the JAXR-based infomodel. Such objects are used in the User JAXR Object.</p>	Method	<p>type (optional) – the type of the telephone numbers.</p>
<p>@ExternalLink</p>	Method	<p>slotName (optional) – the name of a slot inside the ExternalLink registry object to be mapped</p>

Annotations and Description	Scope	Properties
Maps a bean property to a ExternalLink JAXR-based object or a collection of them.		<p>that is checked for having a specified value. This is used to pick the proper ExternalLink if the registry object has more than one.</p> <p>slotValue (optional) – the value of the slot to be checked.</p> <p>type (optional) – type of the bean used for the mapping</p>
<p>@Association</p> <p>Maps a property to an association. It can be either the association object itself or the target of the association.</p>	Method	<p>key (optional) – the key of the association type to be used. Either type or key must be present.</p> <p>type (optional) – the association type to be used. Either type or key must be present.</p> <p>targetType (optional) – the type of the bean to be mapped. It is used when the bean property is a collection and the type cannot be guessed.</p> <p>mappedTo (optional) – the property can be mapped to either the association registry object or the target of the association.</p> <p>cascadeStyle (optional) – Supported cascade styles are ALL (Cascade on all operations), UPDATE (Cascade on update operations), DELETE (Cascade on delete operations), NONE (no cascading).</p>
<p>@AssociationTarget</p> <p>Used in conjunction with the @Association annotation. Maps a bean to a target of an association. It is used when a bean is mapped to an association object using the @Association annotation. Then inside that bean a property must be mapped to the target.</p>	Method	None
<p>@Classification</p> <p>Maps a bean property to a classification. Both the classification object and its concept can be used. The mapping can be simple – Bean property <-> Classification(Concept)</p>	Method	<p>classificationScheme (optional) – the name of the ClassificationScheme to be used.</p> <p>parentConcept (optional) – the path of the parent concept. Used when mapping enumeration classifications.</p>

Annotations and Description	Scope	Properties
or enumeration – Bean property <-> Classification (Concept) which concept is under a specified parent concept. The latter provides a set of predefined possible concepts, thus is similar to the notion of enumeration.		<p>parentConceptKey (optional) – the key of the parent concept. Either the path or the key can be used.</p> <p>conceptPath (optional) – the path of the concept for this classification.</p> <p>conceptKey (optional) – the key of the concept for this classification. Either the path or the key can be used.</p> <p>targetType (optional) – the type of the bean used for the mapping. Required when the property is a collection and the type cannot be guessed.</p> <p>mappedTo (optional) – the bean can be mapped either to the classification object or to its concept.</p> <p>cascadeStyle (optional) – The supported cascade styles are ALL, UPDATE, DELETE, and NONE.</p>
<p>@ClassificationConcept</p> <p>Used in conjunction with the @Classification annotation. Maps a bean to the Concept of the Classification specified in the @Classification annotation.</p>	Method	None
<p>@ClassifiedInstances</p> <p>Maps class hierarchy to registry objects. Classifications are used to achieve this. Each registry object that corresponds to a bean from the hierarchy is classified with a concept. The latter belongs to a taxonomy mirroring the class hierarchy.</p>	Type	<p>instances (mandatory) – the array of the instances that this mapping will address.</p>
<p>@ClassifiedInstance Sets the information for a specific mapping between a bean from the hierarchy and a registry object.</p>	Type	<p>classificationScheme (mandatory) – the classification scheme to which the concept belongs. Either the scheme name or the key must be specified.</p> <p>classificationSchemeKey (mandatory) – the key of the classification scheme. Either the scheme name or the key must be specified.</p>

Annotations and Description	Scope	Properties
<p>@ClassificationAttribute</p> <p>Annotation for mapping the return value of a (getter) method to the classification attribute specified at type level. The attribute name is mandatory and is used to identify the attribute. This annotation is very similar to the <code>{@link Classification}</code> annotation in terms of supported attributes and underlying representation. The difference is that the taxonomy is obtained from the attribute description. In order to use this annotation, a classification attribute must be defined at type level (the registry object type must have a classification attribute with the same attribute name as specified in the annotation).</p>	Method	<p><code>conceptKey</code> (mandatory) – the key of the concept used to classify this instance.</p> <p><code>conceptPath</code> (mandatory) – the path of the concept used to classify this instance.</p> <p><code>beanType</code> (mandatory) – the type of the bean that corresponds to this instance.</p>
		<p><code>attributeName</code> (mandatory) – The name of the attribute represented by this annotation.</p> <p><code>cascadeStyle</code> (optional) – The cascading style for this mapping.</p>
		<p><code>targetType</code> (optional) – The type of the mapped bean. The bean itself must be of type <code>Concept</code>.</p>
<p>@FileAttribute</p> <p>Annotation for mapping the return value of a (getter) method to the file attribute specified at type level. The attribute name is mandatory and is used to identify the attribute. This annotation is very similar to the <code>{@link ExternalLink}</code> annotation in terms of supported attributes and underlying representation. In order to use this annotation, a file attribute must be defined at type level (the registry object type must have a file attribute with the same attribute name as specified in the annotation).</p>	Method	<p><code>attributeName</code> (mandatory) – The name of the attribute represented by this annotation.</p> <p><code>cascadeStyle</code> (optional) – The cascading style for this mapping.</p> <p><code>targetType</code> (optional) – The type of the mapped bean. The bean itself must be of type <code>ExternalLink</code>.</p>

Annotations and Description	Scope	Properties
<p>@Relationship</p> <p>Annotation for mapping the return value of a (getter) method to the attribute specified at type level. The attribute name is mandatory and is used to identify the attribute. This annotation is very similar to the {@link Association} annotation in terms of supported attributes and underlying representation. The difference is that the association and target types are not specified but are obtained from the attribute description. In order to use this annotation, a relationship attribute must be defined at type level (the registry object type must have a relationship attribute with the same attribute name as specified in the annotation).</p>	Method	<p>attributeName (mandatory) – The name of the attribute represented by this annotation.</p> <p>cascadeStyle (optional) – The cascading style for this mapping.</p> <p>targetType (optional) – The type of the mapped bean. The bean itself must be of type Concept.</p>

Example

```

/**
 * Java bean interface representing JAXR-based registry objects
 * of type ServiceInterfaceVersion.
 */
@RegistryObject(objectTypeName =
"{http://namespaces.CentraSite.com/csaf}ServiceInterfaceVersion")
@Bean(implementationClass =
"com.softwareag.centrasite.appl.framework.persistence.beanmodel.impl.ServiceInterfaceVersionImpl")
public interface ServiceInterfaceVersion extends
    RegistryBean{

    @Property(target = "name")
    public String getName();
    public void setName(String name);

    /**
     * Returns the description
     */
    @Property(target = "description")
    public String getDescription();

    /**
     * Sets the description
     */
    public void setDescription(String description);

    /**
     * Returns the attachments

```

```
    */
    @ExternalLink(type = com.softwareag.centrasite.appl.framework.persistence.
beanmodel.ExternalLink.class) public List<com.softwareag.centrasite.appl.
framework.persistence.beanmodel.ExternalLink> getAttachments();

    /**
     * Sets the attachments
     */
    public void setAttachments(List<com.softwareag.centrasite.appl.
framework.persistence.beanmodel.ExternalLink> attachments);

    /**
     * Returns the short name of the interface version.
     * Maps to {http://namespaces.CentraSite.com/csaf}shortName slot.
     */
    @Slot(name = "{http://namespaces.CentraSite.com/csaf}shortName")
    String getShortName();

    /**
     * Sets the short name property of the interface version.
     */
    void setShortName(String shortName);

    /**
     * Returns.
     */
    @Association(type = "HasReviewRequest",
        targetType = ReviewRequestOutcome.class,
        cascadeStyle = CascadeStyle.DELETE)
    List<ReviewRequestOutcome> getReviewRequestOutcomes();

    /**
     * @param list
     */
    public void setReviewRequestOutcomes(List<ReviewRequestOutcome> list);

    /**
     * Returns the findings, which are attached to the bean.
     */
    @Classification(classificationScheme = "CSAF -Taxonomy",
        conceptPath = "/ClassificationInstances/Finding",
        targetType = Finding.class)
    List<Finding> getFindings();

    /**
     *
     * @param pFindings
     */
    public void setFindings(List<Finding> pFindings);

    @Slots(targetType = SlotBean.class)
    public Collection<SlotBean> getSlots();

    public void setSlots(Collection<SlotBean> slots);
}

/**
 * Implementation of the {@link ServiceInterfaceVersion} bean interface.
 */public class ServiceInterfaceVersionImpl extends DynamicRegistryBean
```

```

implements ServiceInterfaceVersion {

    private String _shortName;
    private List<ReviewRequestOutcome> _reviewRequestOutcomes;
    private Collection<SlotBean> slots;
    private String _instanceSlotName;
    private List<Finding> findings;
    private List<ExternalLink> externalLinks;

    /**
     * {@inheritDoc}
     */
    public String getShortName() {
        return _shortName;
    }

    /**
     * {@inheritDoc}
     *
     * The setter is annotated that modifies the object and it needs to be
     * updated in the JAXR-based registry.
     */
    public void setShortName(String shortName) {
        _shortName = shortName;
    }

    public List<ReviewRequestOutcome> getReviewRequestOutcomes() {
        return _reviewRequestOutcomes;
    }

    public
    void setReviewRequestOutcomes(List<ReviewRequestOutcome> list) {
        _reviewRequestOutcomes = list;
    }

    public Collection<SlotBean> getSlots() {
        return slots;
    }

    public void setSlots(Collection<SlotBean> slots) {
        this.slots = slots;
    }

    public String getInstanceSlotName() {
        return _instanceSlotName;
    }

    public void setInstanceSlotName(String slotName) {
        _instanceSlotName = slotName;
    }

    public List<Finding> getFindings() {
        return findings;
    }

    public void setFindings(List<Finding> findings) {
        this.findings = findings;
    }

    public List<ExternalLink> getAttachments() {

```

```
        return externalLinks;
    }

    public void setAttachments(List<ExternalLink> attachments) {
        externalLinks = attachments;
    }
}
```

Standard Mappings

The Standard Mappings (`com.softwareag.centrasite.appl.framework.beans.standard`) are RegistryBeans that represent all supported JAXR-based Registry Objects under the package `com.centrasite.jaxr.infomodel`. They provide the functionality to operate and manage JAXR-based RegistryObjects through the Application Framework with ease.

There are other kinds of objects that are included in this package although they are not RegistryObjects (`EmailAddress`, `PostalAddress`, `Slot` ... etc.). The Application Framework provides a mapping for them as well. Standard Mapping instances are created by the BeanPool's `create(beanClass)`; standard non-registry object mappings (`EmailAddress`, `PostalAddress`, `Slot` ... etc.) are managed using the following:

```
com.softwareag.centrasite.appl.framework.beans.standard.StandardMappingManager
```

Standard Mappings Usage Sample

```
//Create a com.softwareag.centrasite.appl.framework.beans.standard.Organization
com.softwareag.centrasite.appl.framework.beans.standard.Organization
organization = beanPool.create(com.softwareag.centrasite.appl.framework.beans.s
tandard.Organization.class);
organization.setName("MyOrganization");

// Create StandardMappingManager for managing Standard non RegistryObjects
// mappings
StandardMappingManager smm = new StandardMappingManager(registryProvider);

//Create a postal address
com.softwareag.centrasite.appl.framework.beans.standard.PostalAddress pa =
    smm.createPostalAddress("streetNumber", "street", "city",
                           "stateOrProvince", "country", "postalCode","type");
organization.setPostalAddress(pa);

// Get existing user and add it to the organization
com.softwareag.centrasite.appl.framework.beans.standard.User user =
beanPool.read( com.softwareag.centrasite.appl.framework.beans.standard.
User.class,
USER_KEY);
Collection<User> users = new ArrayList<User>();
users.add(user);
organization.setUsers(users);

// save the changes
beanPool.flush();
```

Generating Beans Through the Command Line

Pre-requisites:

CentraSite provides a command tool named `GenerateCSAFBeans` for this purpose.

The syntax is of the format:

```
C:\SoftwareAG\CentraSite\utilities>GenerateCSAFBeans.cmd <USERNAME> <PASSWORD>
<CENTRASITE-URL> <TYPENAME> <INTERFACEPACKAGE> <IMPLPACKAGE> <DESTINATION>
```

The input parameters are:

Parameter	Description
CENTRASITE-URL	(Optional). The URL of the CentraSite registry. For example, <code>http://localhost:53307/CentraSite/CentraSite</code> .
USER-ID	The user ID of a registered CentraSite user who has the CentraSite Administrator role. For example, <code>Administrator</code> .
PASSWORD	The password for the registered CentraSite user identified by the parameter <code>USER-ID</code> .
TYPENAME	<p>The namespace or name of the type to be generated. Example: <code>{http://test}TestService</code>.</p> <p>Or, the name of the virtual type to be generated. Example: <code>"Virtual service"</code>.</p> <div style="background-color: #f0f0f0; padding: 5px;"> <p>Note: The quotation marks are necessary, in order that <code>"Virtual service"</code> is parsed as a single token.</p> </div>
INTERFACEPACKAGE	The name of the package in which the interfaces should be generated. For example: <code>com.sag.generated</code>
IMPLPACKAGE	The name of the package in which the implementation should be generated. For example: <code>com.sag.generated.impl</code>
DESTINATION	The location where the generated bean will be stored.

Example (all in one line):

```
C:\SoftwareAG\CentraSite\utilities>GenerateCSAFBeans.cmd Administrator manage
http://localhost:53307 "Virtual service" com.sag.generated com.sag.generated.impl c:\tmp\
```

Querying the Registry

The Application Framework provides two search functionalities:

- The [Application Framework Simple Search](#) uses only framework-specific data, so it is simpler to use and supports all needed query operations. This search interface is also the recommended one to use.
- The [Application Framework JAXR-Based Search](#) combines framework and JAXR-based data. The advantage of this search is that it can use the whole JAXR-based functionality to query

the registry. The disadvantage is that in order to use it, the user must have considerable knowledge of JAXR.

Application Framework Simple Search

The Application Framework Simple Search uses framework-specific data only. To perform a search, you:

1. Create a search object using a BeanPool instance.
2. Restrict search results by adding search predicates.
3. Define the order of the search results using one of the `Order` static methods.
4. Invoke the search using the `result` method.

Creating a Search Object

To search the registry, the user must create a search object using a BeanPool instance. The BeanPool offers several methods for creating search objects:

■ Without arguments:

```
BeanPool.createSearch();
```

This creates a search object which, when executed, searches for objects in the registry from all registered bean types. See [“Bean Types Managed by CSAF” on page 237](#).

■ When a List of items is passed:

```
BeanPool.createSearch(List<Class<? extends RegistryBean>> beanClasses)
```

The created search object searches through all objects in the registry, from the specified list of types.

■ When a single type is passed:

```
BeanPool.createSearch(Class<? extends RegistryBean> beanClass)
```

The search object searches the registry only for items from the specified type.

The search object has a `result()` method that searches the registry and returns a list of all `RegistryBean` objects that satisfy the search criteria.

Example:

```
BeanPool beanPool = sessionContext.getCurrentBeanPool();
```

```
Search search = beanPool.createSearch();
```

Restricting the Search Results by Adding Search Predicates

The predicate is an object representation of a query criterion used to restrict the search results. Predicates can be created from a factory-like class called `Predicates` (`com.softwareag.centrasite.appl.framework.persistence.search.Predicates`).

It provides two static methods for creating each specific predicate:

■ **Without specifying Bean Type:**

```
Predicates.eq(String propertyName, Object value)
```

■ **By specifying a Bean Type:**

```
Predicates.eq(String propertyName, Object value,
Class<? extends RegistryBean> beanType)
```

where

<i>method name</i>	The comparison operator:	
	<i>and</i>	logical conjunction
	<i>eq</i>	equal
	<i>ge</i>	greater than or equal to
	<i>gt</i>	greater than
	<i>le</i>	less than or equal to
	<i>like</i>	matches a string that can include wildcards
	<i>lt</i>	less than
	<i>ne</i>	not equal
	<i>or</i>	logical disjunction
<i>property name</i>	<p>The name of the property to be compared. This property name is a string value representing the name of the Java property (<code>getName()</code> corresponds to "name"). The search functionality supports adding a sequence of properties. This is accomplished by knowing the searched <code>RegistryBean</code> property hierarchy and by separating following properties with a dot ..</p> <p>Example:</p> <pre>Predicates.eq("externalLink.uri ", value)</pre> <p>The predicate is created for the URI property of the externalLinks of the searched <code>RegistryBean</code>, which should be equal to the given value.</p>	
<i>value</i>	<p>The value to compare against. Most methods expect an <code>Object</code> value because the search can handle a variety of objects including <code>String</code>, <code>Number</code>, <code>Date</code>, <code>Calendar</code>, <code>Key</code>, <code>RegistryBean</code>, and others. There are also methods that expect a specific value type. An example is <code>like (String propertyName, String value)</code>, which supports wildcards and therefore the</p>	

expected value type is String. Other object types that are worth mentioning are the so-called support types (TelephoneNumbers, InternationalString, LocalizedString, EmailAddress, PostalAddress). They can be used for search criteria but not as a searched object because they are not registry beans. For example, the following search is valid:

```
Search search = beanPool.createSearch(User.class);
Predicates.eq("telephoneNumbers.countryCode",
"someCountryCode");
```

But the following search is not valid:

```
Search search = beanPool.createSearch(EmailAddress.class);
```

beanType

The bean type for which the predicate is applied.

Important:

If no *beanType* is specified then the predicate is applied to the first bean type in the Search object's list of bean types. Note that the first item of that list must support the property passed to the predicate, otherwise the search fails. In cases where the search object is created for all supported bean types, the list is filled randomly so the user must be aware of all common properties supported by these RegistryBean types.

Each predicate can be added to the search object by invoking the search method:

```
addPredicate(Predicate predicate);
```

A search object can add multiple predicates, which can be treated as predicates joined by an and operator. For example:

```
Search search = beanPool.createSearch();
search.addPredicate(predicate1);
search.addPredicate(predicate2);
search.addPredicate(predicate3);
```

is equal to predicate1 and predicate2 and predicate3 in the query to be executed.

There are two more methods in the Predicates class: `and(Predicate p1, Predicate p2)` and `or(Predicate p1, Predicate p2)`. These methods create a so-called combine predicate. They join two predicates by logical conjunction or logical disjunction respectively. This predicate can be added to the search object in the same way as the common predicates explained above.

Supported predicates description

All supported predicates are created from methods in the Predicates class (`com.softwareag.centrasite.appl.framework.persistence.search.Predicates`).

Like Predicate

A predicate that supports usage of wildcards. The value field of the creating methods:

```
like(String propertyName, String value)
like(String propertyName, String value, Class<? extends
RegistryBean> beanType)
```

is of Type String, so the user may add strings (possibly including wildcards).

Example:

```
like("name", "%partOfExpectedName");
```

Wildcards

The like predicate supports wildcards in the manner of SQL and UDDI. The wildcard characters are as follows:

Wildcard character	Indicates
%	Any value for any number of characters
_	Any value for a single character

The following special cases are supported:

To represent...	use the character string...
%	\%
_	_
\	\\

Greater Than Predicate

A predicate that compares Number, Date, or Calendar, returning true if the compared object value is greater than the value given in the predicate's creating method "value" field:

```
gt(String propertyName, Object value)
gt(String propertyName, Object value, Class<? extends
RegistryBean> beanType)
```

The value must be one of the following types: Number, Date, Calendar.

Example:

```
Calendar calendar = Calendar.getInstance();
Predicate predicate = Predicates.gt("requestDate",calendar);
```

Less Than Predicate

A predicate that compares Number, Date, or Calendar, returning true if the compared object value is less than the value given in the predicate's creating method "value" field:

```
lt(String propertyName, Object value)
lt(String propertyName, Object value, Class<? extends
RegistryBean> beanType)
```

The value must be one of the following types: Number, Date, Calendar.

Example:

```
Predicate predicate = Predicates.lt("copyNumber",203);
```

Greater or Equal Predicate

A predicate that compares Number, Date or Calendar, returning true if the compared object value is greater than or equal to the value given in the predicate's creating method "value" field:

```
ge(String propertyName, Object value)
ge(String propertyName, Object value, Class<? extends
RegistryBean> beanType)
```

The value must be one of the following types: Number, Date, Calendar.

Example:

```
Predicate predicate = Predicates.ge("copyNumber",203);
```

Less or Equal Predicate

A predicate that compares Number, Date or Calendar, returning true if the compared object value is less than or equal to the value given in the predicate's creating method "value" field:

```
le(String propertyName, Object value)
le(String propertyName, Object value, Class<? extends
RegistryBean> beanType)
```

The value must be one of the following types: Number, Date, Calendar.

Example:

```
Predicate predicate = Predicates.le("copyNumber",203);
```

Equal Predicate

A predicate that returns true if the compared object value is equal to the value given in the predicate's creating method "value" field:

```
eq(String propertyName, Object value)
eq (String propertyName, Object value, Class<? extends
RegistryBean> beanType)
```

The value must be one of the following types: Number, Date, Calendar, String, Key, RegistryBean.

If the value is of type RegistryBean then the comparison is made by the RegistryBean's key.

Example:

```
Predicate predicate = Predicates.eq("name","somePropertyname");
```

Not Equal Predicate

A predicate that returns true if the compared object value is not equal to the value given in the predicate's creating method "value" field:

```
ne(String propertyName, Object value)
ne (String propertyName, Object value, Class<? extends
RegistryBean> beanType)
```

The value must be one of the following types: Number, Date, Calendar, String, Key, RegistryBean.

If the value is of type RegistryBean then the comparison is made by the RegistryBean's key.

Example:

```
Predicate predicate = Predicates.ne("name","somePropertyname");
```

AND Predicate

A predicate that joins two predicates in a logical conjunction. The method that creates this predicate:

```
public static Predicate and(Predicate p1, Predicate p2)
```

expects two predicates as arguments.

Example:

```
Predicate predicate1 = Predicates.eq("name","somePropertyname");
Predicate predicate2 = Predicates.eq("name","somePropertyname2");
Predicate andPredicate = Predicates.and(predicate1, predicate2);
```

OR Predicate

A predicate that joins two predicates in a logical disjunction. The method that creates this predicate:

```
public static Predicate or(Predicate p1, Predicate p2)
```

expects two predicates as arguments.

Example:

```
Predicate predicate1 = Predicates.eq("name","somePropertyname");
Predicate predicate2 = Predicates.eq("name","somePropertyname2");
Predicate orPredicate = Predicates.or(predicate1, predicate2);
```

Defining the Order of the Search Results

You can define the order using one of the following `Order` (`com.softwareag.centrasite.appl.framework.persistence.search.Order`) static methods, which create ascending or descending order for a given property:

- `asc(String propertyName)` for ascending
- `desc(String propertyName)` for descending

The rules for the property name when creating `Order` are the same as when creating a `Predicate`. The user must know whether the bean types added to the search object support the property passed to the `Order asc(String propertyName)` or `desc(String propertyName)` methods. You can add multiple orders to the search object.

Example:

```
Order order = Order.asc("description");
```

Invoking the Search

After adding the necessary predicates and orders to the search object, the search can be executed by invoking the `result()` method on the search object. It returns a list of all `RegistryBean` objects in the registry that applied the predicate conditions in the specified order. The result is lazy loading compatible.

Here is an example of a Search lifecycle:

```
List searchTypes = new ArrayList();
searchTypes.add(ReviewRequestOutcome.class);
searchTypes.add(ServiceInterfaceVersion.class);

Search search = beanPool.createSearch(searchTypes);

Predicate predicate1 = Predicates.eq("ExternalLink.URI",
"http://www.softwareag.com");
Predicate predicate2 = Predicates.eq("name","somePropertyname2");
Predicate orPredicate = Predicates.or(predicate1, predicate2);

Search.addPredicate(orPredicate);

search.addOrder("name");

List<RegistryBean> result = (List<RegistryBean>) search.result();
```

This means that all `ReviewRequestOutcomes` and `ServiceInterfaceVersions` are searched and the ones that have name equal to “somePropertyname2” or `ExternalLink` with URI equal to “http://www.softwareag.com” is returned in the resulting `List` of `RegistryBean` objects ordered by name.

Extending the Application Framework

There are several points where the user can extend the existing Application Framework functionality.

Properties

Each Java bean property is internally represented as a `com.softwareag.centrasite.appl.framework.mapping.Property` instance.

The recommended way of creating a new property is by extending, directly or indirectly, the `BaseProperty` class (`com.softwareag.centrasite.appl.framework.mapping.BaseProperty`).

To map the information from a given annotation to the new `Property` correctly, a user-defined `Property Processor` that implements the `PropertyAnnotationProcessor` (`com.softwareag.centrasite.appl.framework.PropertyAnnotationProcessor`) must be created.

Then the newly created `PropertyProcessor` must be added to the list of processors in the `BeanTypeAnnotationProcessor` (`com.softwareag.centrasite.appl.framework.BeanTypeAnnotationProcessor`) using the `addAnnotationProcessor(Class<?> annotationType, PropertyAnnotationProcessor annotationProcessor)` method.

Property Mapper

Each property value must be transferred to/from the underlying registry object. For that purpose, CSAF provides the (`com.softwareag.centrasite.appl.framework.persistence.mapper.PropertyMapper`) interface.

Users can provide their own implementation of the `PropertyMapper` interface by hooking it to a given type of `Property`. Such a property mapper is registered using the `com.softwareag.centrasite.appl.framework.persistence.mapper.PropertyMapperFactory.addHandler` (`PropertyMapperFactory.Handler`) method.

Predicate

The preferred method of creating a custom-defined predicate is to extend the `DefaultPredicate` (`com.softwareag.centrasite.appl.framework.persistence.search.impl.DefaultPredicate`) class directly or indirectly. Another way is to directly implement the `Predicate` interface (`com.softwareag.centrasite.appl.framework.persistence.search.Predicate`), although this is not recommended because it does not offer default behavior.

To use this newly-created predicate, the user must create a custom defined predicate handler, which must implement the `PredicateHandler` interface (`com.softwareag.centrasite.appl.framework.persistence.search.PredicateHandler`). This predicate handler must be added to the `PredicateFactory` (`com.softwareag.centrasite.appl.framework.persistence.search.impl.PredicateFactory`) list of predicate handlers by calling `addPredicateHandler(PredicateHandler handler)`.

Application Framework JAXR-Based Search

Whereas the `BeanPool` interface takes care of the standard CRUD operations to the registry, the queries are performed using the `Query` interface (`com.softwareag.centrasite.appl.framework.persistence.Query`):

```
package com.softwareag.centrasite.appl.framework.persistence;
public interface Query<T extends RegistryBean> {
    List<T> run(QueryContext pContext) throws JAXRException,
        CSAppFrameworkException;
```

```
}
```

In order to do a query, one should implement this interface and place the querying routines in the `run()` method implementation. The query is then executed via `BeanPool.run()`:

```
<T extends RegistryBean> List<T> run(Query<T> pQuery)
    throws CSAppFrameworkException;
```

The returned data is then in the form of beans.

This mechanism still requires knowledge of JAXR. The benefit is that JAXR is isolated in this interface. Sample implementation of Query:

```
final Query<EntryCode> q = new Query<EntryCode>() {
    public List<EntryCode> run(QueryContext context) throws JAXRException {
        final RegistryAccessor regDAO = context.getRegistryAccessor();
        final Concept concept = regDAO.findConceptByPath("CSAF-Taxonomy",
            "/ClassificationInstances/EntryCodeType");
        final List<EntryCode> result = new ArrayList<EntryCode>();
        for (Concept c : (Collection<Concept>) concep.getChildrenConcepts()) {
            try {
                EntryCode ec = context.getCurrentBeanPool().read(EntryCode.class,
                    c.getKey().getId());
                result.add(ec);
            } catch (Exception e) {
                throw new RuntimeException(e.getMessage(), e);
            }
        }
        return result;
    }
};
List<RegistryBean> queryResult = getBeanPool().run(q);
```

In general, a Query would use the JAXR-based API to find and retrieve the data, and then the keys of registry objects that were found are passed to the BeanPool to build the beans. These beans are then returned as the result of the query execution.

Event Mechanism

The CSAF allows the user to register and receive notifications when certain events occur. Currently, three persistence events are supported: `objectDeleted`, `objectCreated`, `objectUpdated`. These events can be intercepted by implementing the interface `com.softwareag.centrasite.appl.framework.persistence.PersistenceEventListener`. Such listeners are registered via the BeanPool, which has methods for adding, removing, and retrieving listeners.

All of the supported events are post events, in other words, they are fired after an action has been performed.

Using CSAF in pre-action events has some limitations. This is because the CSAF tries to establish its own connection to the registry data. Under certain circumstances, it may happen that a user searches the registry for a transient object (for example, an object which is still not persisted into the database) and, on which a pre-action event is executing, in such case the user may not be able to retrieve the transient object created using another connection. As a best practice, we recommend that you use the JAXR-based API connection for any pre-action events.

Asset Types

Type Management provides CRUD (create, read, update, and delete) operations for custom object types. CSAF provides its own classes describing object (asset) types and their attributes. Type Management supports operations on the following attributes: `file`, `classification`, `relationship` and `slot`, where `slot` can be one of the following types:

- `xs:boolean`
- `xs:dateTime`
- `xs:date`
- `xs:time`
- `xs:duration`
- `xs:anySimpleType`
- `xs:integer`
- `xs:string`
- `xs:anyURI`
- `xs:double`
- `xs:decimal`

Type Management also provides CRUD operations for profiles, functionality to associate attributes with profiles, and attach profiles to types. A manager interface `com.softwareag.centrasite.appl.framework.types.TypeManager` is the entry point for the application that uses CSAF.

Note:

In order to use Type Management functionality, the `StandaloneRegistryProvider` instance must be created with the `browserBehaviour` flag set to `true`.

Usage Sample for Type Management

```
private String TYPE_LOCAL_NAME = "TypeLocalName";

private String TYPE_NAMESPACE = "http://test.namespace.test";

private String TYPE_NAME = "{" + TYPE_NAMESPACE + "}"
                        + TYPE_LOCAL_NAME;

// Get a sessionContext instance
SessionContext sessionContext = initSessionContext();

// Get a TypeManager instance from sessionContext
TypeManager typeManager = sessionContext.getTypeManager();

// Create a custom object type
TypeDescription typeDescription = typeManager.createType("TypeDisplayName",
```

```
        "TypeDescription",    TYPE_LOCAL_NAME, TYPE_NAMESPACE);

// Create a Classification Attribute
AttributeDescription attrClass = typeManager.createClassificationAttribute(
    "ClassificationAttributeName", "ClassificationAttributeDescription",
    Constants.CLASSIFICATION_SCHEME_PRODUCTS);

// Add attribute to custom type
typeDescription.addAttribute(attrClass);

// Create Profile
Profile profile = typeManager.createProfile("ProfileName");

// Create a File Attribute
AttributeDescription attrFile = typeManager.createFileAttribute(
    "nameFileAttribute", "descriptionFileAttribute");

// Add attribute to profile
profile.addAttribute(attrFile);

// Add profile to custom type
typeDescription.addProfile(profile);

// Save custom type
typeManager.saveType(typeDescription);

// Get custom type by name
TypeDescription type = typeManager.getType(TYPE_NAME);

// Delete custom type
typeManager.deleteType(type);
```

Association Types

In general, registry objects can be related to each other via associations. An association belongs to a specified association type. CentraSite supports predefined association types, such as `HasParent` and `Uses`; in addition, you can create custom association types.

In CentraSite, an association type is uniquely identified by its value (for example: `HasParent`, `Uses`, etc.). The value is specified when the association type is created, it cannot be subsequently modified.

An association type can optionally have one or more locale-specific display names. If no locale-specific display names are specified, the association type's value is used by default.

Each association type has a forward label, this is shown, for example, when a corresponding association is displayed by the impact analysis.

You can optionally specify a backward label. Multiple association types can share forward and backward labels.

The CentraSite Application Framework type management feature provides methods for creating, updating, deleting, and finding association types.

Usage Sample for Association Type Management

```
// Get a sessionContext instance
SessionContext sessionContext = initSessionContext();
```

```
// Get a TypeManager instance from sessionContext
TypeManager tm = sessionContext.getTypeManager();

AssociationType at = tm.createAssociationType(
    "MyAssociationType", "MyDisplayName", "MyForwardLabel",
    "MyBackwardLabel", Locale.EN);
tm.saveAssociationType(at);

// find an association type by its value
AssociationType myAssociationType = tm.getAssociationType("MyAssociationType");

// find an association type by its display name
myAssociationType = tm.getAssociationTypeByName("MyDisplayName");

// add a display name with a different locale
myAssociationType.setName("MonNom", Locale.FRENCH);
tm.saveAssociationType(myAssociationType);

// delete an association type
tm.deleteAssociationType(myAssociationType);
```

Lifecycle Management

The Application Framework supports the Lifecycle Model (LCM) functionality. The LCM provides the ability to define and track the life-cycle of a service and also provides a way to define and enforce policies that govern the path of an asset through the lifecycle. As a result, these policies can be automated or enforced consistently. Using registry beans, we now support lifecycle-aware registry beans.

The definition of an LC Model starts with the definition of an LC Model taxonomy. The state model of an LC Model is a standard state model (deterministic finite automaton, DFA). The model itself is represented as the concepts of the LC Model taxonomy. A taxonomy is not defined for this, so associations are used to represent the state transitions. The states themselves are just concepts within the taxonomy.

In order to create a lifecycle-aware registry bean, the user must create a registry bean that extends `com.softwareag.centrasite.appl.framework.lcm.beans.LifeCycleAware`. Also, the implementation of this registry bean must extend the `com.softwareag.centrasite.appl.framework.lcm.beans.LCAwareDynamicRegistryBean`. This ensures that the registry bean is lifecycle-aware and is ready to use for lifecycle operations.

In order to manage the lifecycle models and states, the LCM Manager must first be initialized:

```
com.softwareag.centrasite.appl.framework.SessionContext
sessionContext = initSessionContext();
com.softwareag.centrasite.appl.framework.lcm.LCMAdminManager
lcmAdminManager = sessionContext.getLCMAdminManager();
```

The `com.softwareag.centrasite.appl.framework.lcm.LCMAdminManager` provides all operations for creating, modifying and deleting LCModels. State models for Lifecycle Management models can theoretically be complex and encompass multiple machines and LCStates.

LCModels are state machines for Lifecycle Management and the state machines may not have any states that cannot be reached. The `com.softwareag.centrasite.appl.framework.lcm.LCModel` provides methods for all operations that can be performed on an LCModel. When the LCModel becomes active, no

changes to the LCModel are possible; instead, a new version of the LCModel can be created using `LCModel.createVersion()`.

The `com.softwareag.centrasite.appl.framework.lcm.LCState` provides access to the LCState and state specific operations.

For more information about the methods and functionality supported by LCModel, check the Javadoc of the framework.

Usage Sample for LCM

```
// initialize SessionContext
SessionContext sessionContext = initSessionContext();

// get the LCAdminManager
LCAdminManager lcmAdminManager = sessionContext.getLCAdminManager();

// Create a LCModel
LCModel lcModel = lcmAdminManager.createLCModel();
lcModel.setDisplayName("DisplayName");
lcModel.setDescription("Description");

// the LCModel must set a standard mapping Organization:
//com.softwareag.centrasite.appl.framework.beans.standard.Organization
lcModel.setOrganization((Organization)organization, false);

// Create LCStates
LCState lcStateA = lcModel.createLCState();
String stateAName = "State A";
lcStateA.setName(stateAName);
lcStateA.setDescription("stateADesc");
Collection<LCState> states = new ArrayList<LCState>();
states.add(lcStateA);

// add LCStates to lcModel
lcModel.addStates(states);

// lcModel must set an initial State
lcModel.setInitialState(lcStateA);

// add the keys of all Types that should be enabled for LCM
Collection<String> typesToBeEnabledForLCM = new ArrayList<String>();
typesToBeEnabledForLCM.add(typeToEnableForLCMKeys);
lcModel.addEnabledTypes(typesToBeEnabledForLCM);

// Save the lcModel using the LCAdminManager
lcmAdminManager.saveLCModel(lcModel);

// Find existing LCModel.
// The result will contain all LCModels (active and inactive)
// that have the corresponding display name.
List<LCModel> listOfModels =
    lcmAdminManager.findLCModelByDisplayName("DisplayName", false);
```

Revision Management

CentraSite versioning capabilities make it possible to create a new version of an object at any point in time. However, the new version is per definition a new object instance which has to go through the whole lifecycle again, firing creation policies, and so on. There is often a demand for versioning capabilities that allow a defined state of the same object to be restored and referenced. Such a defined state is referred to as a checkpoint.

The CSAF interfaces related to versioning are
`com.softwareag.centrasite.appl.framework.persistence.revision.RevisionManager` and
`com.softwareag.centrasite.appl.framework.beans.RevisionBean`.

The CentraSite revisioning feature can be enabled system-wide, which means that every object modification (create/update) of any instance of any type leads to the creation of a checkpoint.

A checkpoint has the following identifying attributes: a minor version number, a label, and a timestamp. The minor version number is incremented each time a checkpoint is created. The label is an optional description that can be used to add information about the change. Also a timestamp that reflects the date of the checkpoint creation is recorded with the checkpoint. The creation of a new checkpoint is recorded in the audit log.

It is possible to reference one specific checkpoint of an object directly and retrieve all of its data as it was at the point in time when the checkpoint was created. This implies that changes made to the object after the checkpoint took place are not reflected in the retrieved checkpoint. Note that the checkpoints provide read-only access to the data; any attempt to update a checkpoint raises an exception. However the current object can be updated.

Reading a bean instance from the registry using `BeanPool.read()` always returns the current (latest) state of an object.

Deleting an object also deletes all of its checkpoints.

It is possible to purge a set of checkpoints to reduce the amount of data consumed by keeping older states of the object.

Note that in order to use the Revision functionality, the `StandaloneRegistryProvider` instance must be created with the browser Behaviour flag set to false.

Usage Sample for Revision Management

```
package com.softwareag.centrasite.appl.framework.persistence.tests;

import java.util.ArrayList;
import java.util.Collection;

import com.softwareag.centrasite.appl.framework.SessionContext;
import com.softwareag.centrasite.appl.framework.beans.RevisionBean;
import com.softwareag.centrasite.appl.framework.beans.standard.Service;
import com.softwareag.centrasite.appl.framework.persistence.BeanPool;
import com.softwareag.centrasite.appl.framework.persistence.revision.RevisionManager;

public class Revisioning {
    private static String checkpointName = "MyLabel";
```

```
public void revisioning() throws Exception {
    SessionContext sessionContext = initSessionContext();
    BeanPool beanPool = sessionContext.getCurrentBeanPool();

    RevisionManager revManager = sessionContext.getRevisionManager();

    //enable the feature if needed
    if (!revManager.isRevisioningEnabled()) {
        revManager.enableRevisioning();
    }

    // create new checkpoint
    Service bean = beanPool.read(Service.class, "uddikey");
    revManager.setCheckpoint(bean, checkpointName);

    // get all checkpoints including the current state object
    Collection<RevisionBean> checkpoints = revManager.getRevisionBeans(bean);

    // restore to the only checkpoint
    Collection<RevisionBean> restoreObjs = new ArrayList<RevisionBean>();
    for (RevisionBean rev : checkpoints) {
        if (rev.isRevision()) {
            restoreObjs.add(rev);
            break;
        }
    }

    revManager.restoreBeans(restoreObjs);

    // delete checkpoints based on label
    revManager.deleteBeans(checkpointName);
}

private SessionContext initSessionContext() {
    //initialize CSAF
    return null;
}
}
```

Multi-User Scenarios

In order to address multi-user scenarios successfully, several aspects of the framework must be noted.

A `SessionContext` is an expensive-to-create, threadsafe object intended to be shared by all application threads. It is created once, usually on application startup, from a `Configuration` instance. A `BeanPool` is an inexpensive, non-threadsafe object that should be used once, for a single request (single unit of work) and then discarded. The `CurrentBeanPoolContext` interface defines the contract for implementations that know how to scope the notion of a current bean pool.

`ThreadLocalCurrentBeanPoolContext`, which maintains current bean pools for the given execution thread, is provided as an example implementation of this interface.

The specification of JAXR does not support transactions or locking. CSAF and CentraSite's implementation extend the API with some locking and transaction capabilities. Here are some points to note:

- Transactions are handled internally and control over them (including isolation, demarcation, and so on.) is not exposed through CSAF. There is only support for bulk operations by using the `BeanPool.delete(java.util.Collection)` and `BeanPool.update(java.util.Collection)` methods. These methods guarantee the atomicity of the performed operation. There is also a `BeanPool.flush()` which performs one bulk operation for the deleted beans and one for the created and updated beans.
- Each modification to a registry bean (`RegistryBean` instance) leads to obtaining an exclusive lock for writing on the whole registry object in the database. This is a pessimistic locking strategy, as the lock is obtained when the object is modified and not when it is actually persisted.
- Whenever a lock on a registry object cannot be obtained (because it is taken by another client), the following exception is thrown:

```
com.softwareag.centrasite.appl.framework.persistence.LockNotAvailableException
```

- The notion of an outdated object denotes a registry object whose database representation has been changed since it was read. This is usually caused by a different client modifying the same instance. Trying to modify an outdated object leads to the following exception:

```
com.softwareag.centrasite.appl.framework.persistence.ObjectOutdatedException
```

CSAF supports automatic re-reading of outdated objects; this forces a re-read of the object from the database before applying the changes.

In general, the application must minimize the time a registry object is kept locked in the database, that is, the time during which there are ongoing modifications on it.

Setting the Classpath

In order to be able to use the CentraSite Application Framework features, the Java classpath must include all the relevant class files. The easiest way to do this is to include all the JAR files that are contained in the folder `redist` (including the subfolder `redist/csaf`). The `redist` folder is typically located at `C:\SoftwareAG\CentraSite\redist` (Microsoft Windows) or `/opt/softwareag/CentraSite/redist` (UNIX).

Examples

The CentraSite Application Framework SDK comes with two examples. One is for the persistence functionality and the other is for the validation functionality.

CRUD Example

The CRUD example demonstrates the abilities of the persistence framework. It shows how the `BeanPool` is initialized, configured, and connected to the registry. Also it shows how CRUD (create, read, update, and delete) operations are performed and queries implemented and executed. It

also includes the bean model and sample mapping of the most commonly used bean relationships and their JAXR-based representation.

8 Importing Objects Using API

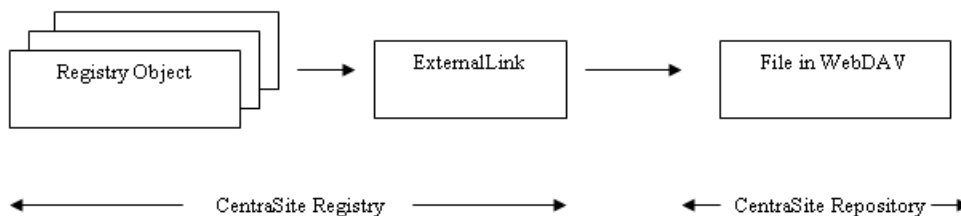
■ Introduction to Importing Objects Using API	266
■ Invoking Importer from Java Program	266
■ Invoking Importer through Command Line	286
■ Invoking Importer Using SOAP API	291
■ Writing Your Own Importer	292

Introduction to Importing Objects Using API

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

- 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 is a registry object (an ExternalLink) that will point to it. Each importer typically creates one or more JAXR-based registry objects. The relationship is:



Invoking Importer from Java Program

The classes used by the importers are available in the `CentraSiteUtils.jar` file. The `CLASSPATH` variable must refer to the JAR files that are used by CentraSite. It is convenient to include all JAR files available 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`.

Importing 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 out of which the organization (per object or per name) parameter must be set while other parameters are optional. If a WSDL imports

includes WSDLs or schemas, then the referenced files are also imported and the according CentraSite objects will be created.

The following example illustrates 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();
}
```

Setter Methods

The `WebServiceRegistrator` class provides the following setter methods. 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 must 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 must be registered.

Syntax

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

Parameters**Organization organizationObject**

The organization object under which the service must be registered.

setPackageName

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

Syntax

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

Parameters**String packageName**

The name of the registry package of which the service must be a member.

Usage Notes

Optional.

setPackageObject

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

Syntax

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

Parameters

RegistryPackage packageObject

The registry package object of which the service must be a member.

Usage Notes

Optional.

setUsedObject

Sets a registry object that will point to the imported service with a `uses` association. That is, the imported service is 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 is pointed to by the imported service with a `uses` association. That is, the imported service uses the specified object.

Syntax

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

Parameters

RegistryObject usesObject

The registry object that is 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 must be stored.

Default: /projects/WSDL.

Usage Notes

Optional.

setUserVersion

Sets a user-defined version number for the service and related registry objects.

Syntax

```
public void setUserVersion (String userVersion)
```

Parameters

String userVersion

The user-defined version identifier for the service objects.

Usage Notes

Optional.

setServiceName

Sets a user-defined display name for the service that must be registered.

Syntax

```
public void setServiceName (String serviceName)
```

Parameters

String serviceName

The display name of the service. The default name is the name of the element <service> defined in the WSDL file.

Usage Notes

Optional.

Importing Web Service from URL

If a WSDL has to be accessed from a protected URL, you can specify its credentials by using a separate constructor of `WebServiceRegistrator`.

Example:

```
WebServiceRegistrator wsr = new WebServiceRegistrator(wsdURL,
    jaxr, null, wsdAccessUser, wsdAccessPassword);
```

Updating 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.

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, however, a new registry entry is created.

Attaching WSDL to Registered Web Service

You can 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 is:

```
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 must be attached.

Note:

- 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 new WSDL, it is removed.
- If the service has a `ServiceBinding` that is in the WSDL, it is updated.

- Operations which are no longer present in the WSDL will be removed.
- Using the `setCreateVersion()` method it is possible to create a new version from the attached service rather than updating it.

The `WebServiceRegistrator` class provides the setter method (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(String attachServiceID)
```

Parameters

String attachServiceID)

The ID of the service, in the form "uddi:...".

setCreateVersion

Indicates if a new version of the service has to be created rather than updating it. This option has effect only if the service is already present.

Syntax

```
public void setCreateVersion (boolean createVersion)
```

Parameters

boolean createVersion

The boolean value of `true` indicates the creation of a new version.

Usage Notes

Optional.

Removing Registered Web Service

This section describes the functions to remove a registered web service, its resources, 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. Close the session by calling the `close()` method of `JAXRAccessor`.

Also before calling the `removeService` or `removeServiceByID` function, call the setter function.

This example demonstrates how to remove a web service from CentraSite:

```
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();
}
```

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	The specified service was not found.

removeServices

Removes all services from CentraSite 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	The specified service was not found.

setDeleteTargetAssocs

Specifies whether 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, then 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 to remove the imported WSDL and schema files together with their controlling ExternalLinks from the repository referenced .

Syntax

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

Parameters

boolean removeReferencedImports

true: Remove the referenced imported WSDL and schema files.

Finding Registered Web Service

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

```
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("MyOrganization".wsdlFile);
    . . .
}
catch (Exception e)
{
    // handle error
}
finally
```

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

The `WebServiceAdministrator` class provides the following methods:

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. Null if all organizations are to be discovered.

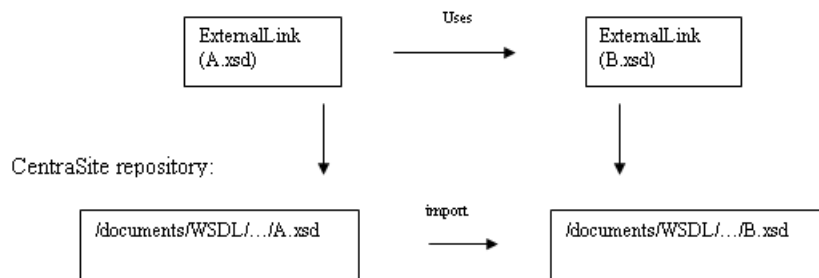
String wsdlFile

The file path to the WSDL file.

Importing Schema

The schema importer works closely 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 the 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. This figure illustrates the relationship:

CentraSite registry:



This 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 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.

This 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 = ...;

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

try
{
    SchemaAdministrator sa = new SchemaAdministrator(jaxr);
    sa.removeByName("MySchema.xsd", true);
}
catch (Exception e)
{
    // handle error
}
finally
{
    jaxr.close();
}
```

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

removeByName

Removes the schema specified by name.

Syntax

```
public int removeByName (String schemaName, boolean cascading)
```

Parameters

String schemaName

The name of the schema registry object.

boolean cascading

true: The specified schema and all its descendant schema objects will be removed.

false: The specified schema will be removed.

Return Codes

The number of schema objects removed.

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 all schemas that are related to it by import and 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 all schemas that are related to it by import and 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 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.

This example demonstrates how to import a BPEL file:

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

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

The BPELRegistrar 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 such as 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

Value	Meaning
true	Each time a service that has not yet been registered in <code>CentraSite</code> is encountered, a counter is incremented. Use the <code>getWarningCount()</code> method to get the current value of the counter.
false	If a service that has not yet been registered in <code>CentraSite</code> is encountered, an exception occurs.

Usage Notes

The default value of the parameter is `true`.

Removing 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 from 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 XPDL File

The XPDL importer imports a process definition from an XPDL file and from the XPDL file, the importer produces a Process object and related components (for example, Process Steps, Process Pools and Process Swimlanes). If the XPDL process references a web service, the importer adds the web service to the registry if it is not already present and associate it with the Process object.

This 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 or 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 uses to
// log on to CentraSite
String user = "";
String password = "";

// 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 or 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. 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 adds 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 any 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 Product Suite, with the subcategories that include:
 - webMethods BPM
 - webMethods Composite Application Framework
 - webMethods Trading Networks

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

Invoking Importer through Command Line

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.
2. Execute the script file with the appropriate input parameters.

Note that there are usable scripts available for most of the importers. You can find the script files in the `utilities` folder in the `<CentraSiteInstall_Directory>`. For example, you could import a Web Service asset using the command tool named `ImportWSDL`. `[cmd|sh]`.

Creating 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.

Creating Script File for Windows (a .bat File)

Create a script file if CentraSite is running under Windows. Example of script file:

```
@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. For a list of the importer class names, see [“Importer Class Names” on page 287](#).

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 Script File for UNIX (Bourne-Shell Script)

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

```

#!/bin/sh
CENTRASITE_HOME=/opt/softwareag/CentraSite
export CENTRASITE_HOME
. ${CENTRASITE_HOME}/bin/centrasite_setenv.sh
# set CLASSPATH
REDIST=${CENTRASITE_HOME}/redist
CLASSPATH="$REDIST:$REDIST/*"
export CLASSPATH
# execute utility
EXECUTABLE="${CS_JAVA_EXE} -cp ${CLASSPATH} importerClassName"
$EXECUTABLE "$@"
RC=$?
exit $RC

```

Where *importerClassName* is the name of the Importer class that you want to run. For a list of the importer class names, see [“Importer Class Names” on page 287](#).

Example

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

```

#!/bin/sh
CENTRASITE_HOME=/opt/softwareag/CentraSite
export CENTRASITE_HOME
. ${CENTRASITE_HOME}/bin/centrasite_setenv.sh
# set CLASSPATH
REDIST=${CENTRASITE_HOME}/redist
CLASSPATH="$REDIST:$REDIST/*"
export CLASSPATH
# execute utility
EXECUTABLE="${CS_JAVA_EXE} -cp ${CLASSPATH}
com.centrasite.jaxr.schema.SchemaImporter"
$EXECUTABLE "$@"
RC=$?
exit $RC

```

Importer Class Names

The table lists the importer class names.

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
REST Service (Importer)	com.centrasite.jaxr.xmlservice.XMLServiceManager
OData Service (Importer)	
XML 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 Script File to Invoke Importer

To invoke the importer, you must run the importer script file with the required set of input parameters. The input parameters that are required to run the script file varies depending on the importer your script file invokes.

Note:

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

Importing Web Service

> To import a Web Service from the command line

- Run your importer script file or use the delivered ImportWSDL script.

The syntax is of the format: `yourScriptFile -dburl centrasiteurl -w wsdlFile -user yourCSUserID -password yourPassword`

To get an overview of all available parameters, call the script without any option. Use the `-help` option to get a detailed description of the parameters.

Example:

```
ImportWSDL -w d:\myDirectory\myWSDLFile.wsdl -o "MyOrganization" -user myUser -password myPassword
```


Invoking the Web Service Administrator from 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 without any option.

Importing REST Service

➤ To import a REST Service from the command line

- Run your importer script file or use the delivered `import Service` script.

The syntax is of the format: *yourScriptFile -s xmlFileURI -n serviceName -e endpointURL -m httpMethods -user yourCSUserID -password yourPassword*

To get an overview of all available parameters, call the script without any option. Use the `-help` option to get a detailed description of the parameters.

Example:

```
import Service -s http://fs02hq/xml/myService.xsd -n myXMLService -e
http://appsrv02:53307/myService -m GET PUT -user myUser -password myPassword
```

Importing XML Schema

➤ To import an XML Schema from the command line

- Run your importer script file or use the delivered `ImportSchema` script.

The syntax is of the format: *yourScriptFile -s xsdFile -user yourCSUserID -password yourPassword*

To get an overview of all available parameters, call the script without any option. Use the `-help` option to get a detailed description of the parameters.

Example:

```
ImportSchema -s d:\myDirectory\myXSDFile.xsd -user myUser -password myPassword
```

Invoking the XML Schema Administrator from 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 without any option.

Importing BPEL Process

➤ To import a BPEL process from the command line

- Run your importer script file or use the delivered `ImportBPEL` script.

The syntax is of the format: *yourScriptFile* -file *bpelFile* -user *yourCSUserID* -password *yourPassword*

To get an overview of all available parameters, call the script without any option. Use the -help option to get a detailed description of the parameters.

Example:

```
ImportBPEL -file d:\myDirectory\myBPELFile.bpel -user myUser -password myPassword
```

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, run the script file without any option.

Importing XPDL File

The XPDL importer imports a process definition from an XPDL file and from the XPDL file, the importer produces a Process object and related components (for example, Process Steps, Process Pools and Process Swimlanes). If the XPDL process references a web service, the importer adds the web service to the registry if it is not already present and associate it with the Process object.

This 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 or 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 uses to
// log on to CentraSite
String user = "";
String password = "";

// 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 or 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.

Invoking Importer Using 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 Software AG Runtime is running and *port* is the port on which Tomcat is listening (port 53307 if CentraSite is configured to use the default Software AG Runtime port number).

Example

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

Software AG recommends using MTOM, 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.



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 `<RuntimeDir> /workspace/webapps`.

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
*_SWT.xml	Files generated for the GUI
images	Directory for icons (GIF files)
lib	Directory for JAR files provided by the plug-in
accesspath	Directory created by HTML generation

The `ImportMyFile` framework 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.

The Build Environment

This topic 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, the Java-based build tool.

Example of the file system structure under the plug-in directory:

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 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.

This is `build.xml`:

```
<?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>

```

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.

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 (`xml/ImportMyFile.xml`) 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>

```

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>

```

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 topic 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.

Here is `src/com/importer/myfile/control/ImportMyFileCommand.java`:

```
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);
    }
}
```

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 (this is `src/com/importer/myfile/control/ImportMyFileAdapter.java`):

```
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 "";
    }
}
}

```

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

The syntax for the command is:

```
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 `<RuntimeWebAppsDir>/PluggableUI/` of the installed CentraSite Control. Thus, for example, `ImportMyFile` must 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 `build.xml` command, which installs `ImportMyFile.zip` into the `<RuntimeWebAppsDir>/PluggableUI/` folder.

The syntax for the command is:

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

Generating Additional Files

You need this step only if you are not able to install the necessary files directly in the `<RuntimeWebAppsDir>/PluggableUI/` directory. In this case, you have to build the structure shown in [“Propagating the Plug-In” on page 300](#) 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.

To generate additional files

1. Edit the file stored in the following directory:
`<RuntimeWebAppsDir>/PluggableUI/cis/config/cisconfig.xml`
2. 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"
```

3. Restart the Software AG Runtime. Now you can start the Application Designer with the following URL: `http://localhost:53307/PluggableUI/HTMLBasedGUI/workplace/ide.html`

4. Navigate to **Tools & Documentation > Layout Manager** and select the ImportMyFile Plugin as Application Project. It appears in the Layout Definitions List.
5. Generate the required files using the Operations button on multiple Items > (Re)Generate HTML pages.

Activating the Plug-In

1. Restart Software AG Runtime.
2. Start CentraSite Control.
3. Select the **Import** function.
4. Select the name of the plug-in and click **Next**.
5. Type the name of the file to be imported.
6. Click **Finish**.

Deactivating the Plug-In

1. Run the command:

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

2. Restart Software AG Runtime.
3. Start CentraSite Control.

9 Setting Up CentraSite Eclipse Plug-ins

■ Installing CentraSite Plug-ins in Your Own Eclipse Environment	304
■ Connecting Eclipse to CentraSite	305
■ Using CentraSite Online Documentation	306

Installing CentraSite Plug-ins in Your Own Eclipse Environment

➤ To install the CentraSite plug-ins in your own Eclipse Environment

1. CentraSite plug-ins require the Eclipse SDK. If you want to use CentraSite Reporting, download the Eclipse IDE for Java and Report Developers and select an integrated package that contains both the required Eclipse software and BIRT.

If you do not want to use CentraSite Reporting, you can download the Eclipse IDE for Java Developers. Software AG recommends using the same Eclipse version that Software AG Designer uses.

2. Install a supported JRE as listed in *System Requirements for Software AG Products*.
3. Start Eclipse.
4. Go to the menu entry **Help** and click the option to install software.
5. In the **Available Software** dialog, select **Add** or **Add Site**.
6. In the **Add Repository** dialog, click **Archive**.
7. Go to the `<Software AG_directory>/Designer/updates` directory, select `com.softwareag.common.zip`, click **Open** and then click **OK**.

If you need a language pack, install it using the same method.

8. On the **Add Repository** dialog, click **Archive**.
9. Go to the `<Software AG_directory>/Designer/updates` directory, select `CentraSiteUpdateSite.zip`, click **Open** and then click **OK**.

If you need a language pack, install it using the same method.

10. The **Available Software** dialog lists the CentraSite plug-ins under SOA Governance. Click **Select All** and then click **Next**.
11. Accept the terms of the license agreements and click **Finish**.
12. Restart Eclipse.
13. Test the installation as follows:
 - a. Go to **File > New > Project > Business Intelligence and Reporting Tools > Report Project** and create a report project.

- b. Go to **File > New > Report** and open the **Report Design** perspective.
 - c. Open the view **Report Design > Data Explorer**. Right-click **Data Sources** and then click **New Data Source**. The dialog box lists the CentraSite data sources you can use to create the XQuery for generating CentraSite reports (that is, CentraSite and CentraSite XQuery).
 - d. Select **CentraSite XQuery**. Use the default data source name and click **Next**. Connection information about the CentraSite data source appears.
14. Install fixes in your Eclipse environment as follows:

Note:

Before installing fixes in your Eclipse environment, you have to install all the fixes in your existing Software AG Designer installation.

- a. Go to **File > Import > Install > From Existing Installation** and click **Next**.
- b. In the **Import from Application** dialog box, browse to the Eclipse plug-ins in your Software AG Designer installation that contains the latest fixes and click **Next**. The Eclipse plug-ins are located at `<Software AG_directory>/Designer/eclipse`.
- c. Select **CentraSite Eclipse Plug-Ins** and **CentraSite Reporting Tools** from the list displayed and install them.

Connecting Eclipse to CentraSite

➤ To connect Eclipse to CentraSite

1. Go to the **Window** menu entry and click **Preferences**.
2. Expand the **CentraSite** node and click **Connections**.
3. Click **Add** and provide the following information:
 - Unique name to appear in your list of connections.
 - Host and port for the Software AG Runtime. The default port is 53307.
 - ID of a user who has permission to access the CentraSite Registry Repository, and password for that user.
4. Click **OK**.
5. Check that a connection to CentraSite is possible by clicking **Test**. If you experience problems, go to **Window > Preferences > General > Network Connections** and check the network connection settings. The proxy settings should be the same as for the Software AG Runtime.

6. Go to **Window > Open Perspective > Other** in the main menu of the Eclipse Workbench and select the CentraSite perspective.

Using CentraSite Online Documentation

The CentraSite Eclipse plug-ins are integrated in the help system of the Eclipse Workbench. To access the help you need an Internet browser that supports JavaScript, Java applets, and Cascading Style Sheets. The documentation has been successfully tested with the Sun JVM 1.8.0 browser plug-in.

If you are using Internet Explorer, when you try to display CentraSite help pages that use active content, you might receive warning messages (for example, To help protect your security, Internet Explorer has restricted this webpage from running scripts or ActiveX controls that could access your computer). Change the browser options to allow active content to run in files on your computer. Then restart Internet Explorer, click the information bar, and click **Allow Blocked Content...** for each affected help page.

- To see *CentraSite Eclipse UI Help*, go to **Help > Help Contents**.
- To see context-sensitive help for a CentraSite Eclipse plug-in view, menu option, dialog, property page or wizard, press F1.