CentraSite Developer's Guide

Version 9.8

April 2015

Document ID: CS-DG-98-20150415

Table of Contents

About this Guide....13
- Document Conventions....13
- Online Information....14

Developing Custom Actions....15
- Planning to Create Custom Actions....16
- About Action Categories....16
  - Predefined Action Categories Installed with CentraSite....16
  - Custom Action Categories....17
- About Action Templates....17
  - Types of Actions....17
  - Supported Object Events....18
- About Parameter Templates....18
- Who Can Create and Manage Action Categories or Templates?....18
- Viewing the Action Categories List....19
- Adding Custom Actions Using the CentraSite UI....19
  - Creating a Custom Action Category....19
  - Adding an Action Template to a Custom Action Category....20
  - Adding a Parameter Template to the Action Template....22
- Adding Custom Actions Using APIs....23
  - Creating Action Rules....23
    - Creating a Rule in a Java Class....23
    - Creating a Rule Using a Groovy Script....24
  - Uploading Action Rules to Action Templates....25
- Viewing or Editing Action Categories or Templates....26
  - Viewing or Editing an Action Category....26
  - Viewing or Editing an Action Template....26
- Downloading Rules from System Action Templates....27
  - Structure of the Zip File....27
- Deleting Custom Action Categories and Templates....27
  - Deleting a Parameter Template....28
  - Deleting a Custom Action Template....28
  - Deleting a Custom Action Category....28
- Versioning a Custom Action Template....29
- Sample Custom Actions....30
  - Sample Java Action: Enforce Unique Asset Names....30
  - Sample Groovy Script Action: Service Attribute Checker....32

Built-In Design/Change-Time Actions Reference....37
- Summary of Actions in the ARIS Category....38
- Summary of Actions in the Change-Time Category....38

Summary of Actions in the Collector Category..40
Summary of Actions in the Design-Time Category..41
Summary of Actions in the Global Category..41
Summary of Actions in the Handler Category..42
Summary of Actions in the WS-I Category..43
Built-In Actions for Design/Change-Time Policies..44
Call Web Service..44
Change Activation State..46
Change Deployment Status...48
Classify...49
Consumer WSDL Generator..50
Default Move Handler...50
Delete RuntimeEvents and RuntimeMetrics...51
Enforce Unique Name...52
Initiate Approval..52
Initiate Group-Dependent Approval...58
Mark Pending on Runtime Policy Change..61
Notify ARIS Service...61
On Consumer Registration Request Send Email to Owner..63
Processing Steps Status...64
Promote Asset...65
Publish to API-Portal...70
Register Consumer..71
Send Email Notification...72
Set Attribute Value..76
Set Consumer Permission..77
Set Instance and Profile Permissions..78
Set Permissions...81
Set Profile Permissions..83
Set State..83
Set View Permission for Service and Service Related Object to Everyone Group............84
Send Email Notification to Watchers...84
UnClassify...85
UnPublish from API-Portal..86
Validate Attribute Value...87
Validate Classification..88
Validate Description..88
Validate Lifecycle Model Activation...89
Validate Name...90
Validate Namespace...90
Validate Policy Activation...91
Validate Policy Deactivation...92
Validate Service Binding..93
Validate State...93
Validate WSDL Size...94

webMethods REST Publish....94

Access via UDDI.... 97
Overview of the UDDI Standard....98
Summary of UDDI Support in CentraSite.... 99
CentraSite UDDI Architecture....100
Overview....100
Client Access via UDDI....100
Localization....101
Modeling of the Node Business Entity....101
Key Generator tModel....101
Node Business Entity....102
Inquiry Service....102
Publish Service....103
Security Service....104
Custody and Ownership Transfer Service.... 105
WSDL.... 105
UDDI Representation of the Object Model....106
Attributes....106
Key/Value Pair Attributes....106
Rich Text Attributes.... 107
Document Attributes....107
Relationship Attributes....108
Metrics Definition....109
Metrics Reference Taxonomy....110
Metrics Types Taxonomy....111
Total Request Count Taxonomy....111
Success Request Count Taxonomy.... 112
Fault Request Count Taxonomy....112
Average Response Time Taxonomy.... 112
Minimum Response Time Taxonomy....113
Maximum Response Time Taxonomy....113
Availability Taxonomy.... 113
Service Liveliness Taxonomy....114
Representing Targets and Target Types.... 114
Representing Status....114
Representing Version.... 114
Mapping WS-PolicyAttachments.... 115
Configuring the UDDI Environment....115
Configuration Properties....115
UDDI in a Multi-CAST Environment....115
CAST Registration/Deregistration.... 115
Registering a CAST....116

Registering a CAST....116

Deregistering a CAST........ 117
Deployment Descriptors........ 118
Changing the User ID/Password of the Web Application Login Account........ 118
Setting Global and Local UDDI Properties........ 118
Setting Global UDDI Properties........ 119
Setting Local UDDI Properties........ 123
Getting Global and Local UDDI Properties........ 125
Schema Validation of UDDI Requests........ 126
Checked Value Set Validation........ 126
Predefined Value Sets........ 126
Predefined tModels........ 127
UDDI V3 APIs........ 129
Overview........ 129
Classes and Interfaces........ 130
RegistryService........ 130
RegistryConfiguration........ 130
RegistryFramework........ 130
RegistryAgent........ 131
UDDI_Security_SoapService........ 131
UDDI_Inquiry_SoapService........ 131
UDDI_Publication_SoapService........ 131
UDDI_Taxonomy_SoapService........ 131
CentraSiteBusinessService........ 131
Examples........ 132
Getting the Value of an Attribute........ 132
Getting the Proxy Services for a Specified Target........ 133
Inquiring about a Business Service........ 133
Publishing a Business Service........ 134
Fetching Taxonomies........ 134
Using Third-Party IDE Tools with CentraSite........ 135
Overview........ 135
Supported IDE Tools........ 135
Specifying the Inquiry, Publish and Security URLs........ 135
WTP Eclipse 1.5.2 Plug-In........ 136
IBM Rational Application Developer 6.0........ 137
Connecting to CentraSite........ 137
Publishing Entities........ 137
UDDI Extensions........ 137
Using WSDL in a UDDI Registry........ 138
Using WS-PolicyAttachment........ 138
Version 1.2 Support........ 138
Supported Policy Subjects........ 138
Referencing Remote Policy Expressions........ 138
Registering Reusable Policy Expressions........ 139
Registering Policies in UDDI Version 3........ 139

tModels to Support UDDI-Based WS-PolicyAttachments.................................. 140
Version 1.5 Support..141
Extending UDDI Publisher API Set to Enable Physical Deletion of tModels.................. 141
Arguments.. 141
Behavior...141
Returns.. 142
Caveats...142

Pluggable Architecture...143
Introduction to CentraSite Control Pluggable Architecture..144
Customizing the Welcome Page.. 145
Introduction.. 145
Technical Implementation of the Welcome Page.. 147
Overview of Java Methods Used.. 147
Screen Component: Welcome Page..147
Screen Component: Widget... 149
Screen Component: Item... 151
Methods Not Related to Screen Components.. 152
Java Interface Hierarchy.. 153
Installing the Customized Welcome Page...154
Stop Software AG Runtime... 154
Updating the plugin.xml Configuration File... 154
Deploying the New Java Classes to the PluggableUI Environment.................................155
Start Software AG Runtime...155
Example of a Customized Welcome Page...155
Location of Demo Files.. 156
Differences Between the Standard Welcome Page and the Demo Welcome Page.. 156
Implementation of Welcome Page Layout... 157
Header Area...157
Separator Between Header Part and Widget Part..158
Widget CentraSite... 158
Widget Useful Links.. 159
Widget User Preferences..161
Default Settings for Widgets.. 162
Implementing the Demo as an Eclipse Java Project.. 162
Building the Deployment Files for Software AG Runtime...163
The build.properties File... 163
Building the Deployment Files... 163
Building the Deployment Files Using Eclipse (Method 1)................................ 164
Building the Deployment Files from the Command Line (Method 2).................164
Deploying the Demo to Software AG Runtime...164
Displaying the Demo Welcome Page.. 165
Special Programming Techniques..165
Customizing Content Pages...167

Extension Points....167
I18N for Layouts....167
Parameters for Plug-ins.... 168
ConnectionHandler - Logon and Logoff / Exit....168
Perspectives.... 169
Topic.... 171
Command for Item.... 172
Bulk Command for Items.... 173
Add Property.... 174
Tab in Detail View.... 175
Add Source of Notification.... 176
Impact Analysis: NodeDecorator....177
Append Root Node to Topic....178
Replace Standard Detail View by Another Editor.... 179
Extend Search Dialog by Additional Conditions....180
Download Documents.... 182
Making the Download Menu Entry Visible/Invisible for Guest Users....182
Changing the Text String Displayed in the Context Menu.... 182
Changing the Format of the Zipped Archive....183
Attach Documents.... 183
Activating the IDE....185
Step-by-Step Guide.... 185
Setting the Preferred Plug-In and Order of Plug-Ins....186
Installing and Uninstalling Plug-Ins.... 186
Directory Structure....186
Installing a Plug-In....187
Uninstalling a Plug-In.... 187
Plug-In Management Perspective.... 188
Special and Advanced Topics.... 189
Icons.... 189
Class Loading....189
Multithreading and Synchronization.... 191
Nested Layouts....191
Javadoc Documentation of the APIs....192
Step-by-Step Guide.... 192
Eclipse Prerequisites.... 192
Setting up the Plug-in Project.... 193
Plugging into CentraSite Control....199
Bring Your Own Layouts to the Screen....203

Application Framework.... 217
Introduction.... 218
RegistryBean.... 219
BeanPool.... 220
StandaloneRegistryProvider.... 220

Configuration....221
Bean Types Managed by CSAF....222
Bean Modes....222
Persistence Modes....222
Cache Configuration....223
Re-Reading Outdated Objects....223
Mapping Beans to Registry Objects with Annotations....223
Introduction to Bean Mapping....223
Standard Mappings....232
Standard Mappings Usage Sample....232
Generating Beans from the Command Line....233
Querying the Registry....234
Application Framework Simple Search....234
Creating a Search Object....235
Restricting the Search Results by Adding Search Predicates....235
Defining the Order of the Search Results....240
Invoking the Search....241
Extending the Application Framework....241
Application Framework JAXR-Based Search....242
Event Mechanism....243
Asset Types....243
Usage Sample for Type Management....244
Association Types....245
Usage Sample for Association Type Management....245
Lifecycle Management....246
Usage Sample for LCM....247
Revision Management....247
Usage Sample for Revision Management....248
Multi-User Scenarios....249
Setting the Classpath....250
Examples....250
CRUD Example....250

API for JAXR....251
Introduction to the CentraSite API for JAXR....252
Creating and Closing a JAXR-based Connection....252
Creating a JAXR-based Connection....252
Closing a JAXR-based Connection....254
Defining a Service....255
Service that Uses Another Service....255
Service with Additional Information....256
Pre-Defined Classification Schemes (Taxonomies)....256
Impact Analysis....258
CentraSite API for JAXR Reference Information....258
Creating User-Defined Objects....258

Direct XQuery Access to the Stored Data 259
Unique Keys 260
Simultaneous Database Access and Locking 260
Caller 262
Semantics of Remove Operations 262
Delete Operation 263
RegistryObject 263
Association 263
AuditableEvent 264
Classification 264
ClassificationScheme 264
Concept 264
ExternalIdentifier 264
ExternalLink 264
Organization 264
RegistryEntry 264
RegistryPackage 265
Service 265
ServiceBinding 265
SpecificationLink 265
User 265
Unsupported Methods 265
Unsupported FindQualifiers 266
Using Wildcards 266
Using Namespaces 266
Method createSlot 266
Caching Considerations 267
JAXR-based Caching Strategy 267
Caching in User Interfaces 267
Dynamically Loaded JAR Files 267
Cache Location 268

API for XQJ 269
Introduction to the API for XQJ 270
What is XQJ? 270
Features of the XQJ Interface 270
Working with the XQJ Interface 271
Executing an XQuery with a Standard XQExpression 271
Executing an XQuery with an XQPreparedExpression 273
Working with a Materialized XQSequence 275
Examples 275
CentraSite-Specific Extensions to XQJ 275
Updating a Database Using XQJ 276
Inserting a Document in the Registry/Repository 276
XQDataSource Properties 276

Java Management Interface........279
Introduction the Java Management Interface........280
Description........280
Attributes and Operations........281
Attributes........281
Registry/Repository Start/Stop Operations........282

Web Service Interfaces........285
Introduction to the Web Service Interfaces........286
Approval Service........286
About the Approval Service........286
Invoking Operations from the Approval Service........287
Specifying the Authenticated User........287
Specifying the Location of the Approval Log........287
Retrieving the List of Approval Requests that a User Has Submitted........287
ApprovalRequestList Message........288
Getting Details about the Actions of the Approvers Associated with a Request........288
Approving or Rejecting Approval Requests........289
Scrolling Through the List of Returned Approval Requests........290
Reverting a Pending Approval Request........291
Operations........291
getPendingApprovals........291
getApprovalRequests........293
getApprovalActions........295
approve........296
reject........296
getApprovalHistory........297
revertPendingStateChange........299
ApprovalRequestList........300
ApprovalActionResult........302

About this Guide

This guide describes how you can use programming interfaces to access and/or 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 storage locations for services on webMethods Integration Server, using the convention *folder.subfolder:service* .
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).
Italic	Identifies variables for which you must supply values specific to your own situation or environment. Identifies new terms the first time they occur in the text.
`Monospace font`	Identifies text you must type or messages displayed by the system.
{ }	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

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at http://documentation.softwareag.com. The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

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.

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.

1 Developing Custom Actions

- Planning to Create Custom Actions 16
- About Action Categories 16
- About Action Templates 17
- About Parameter Templates 18
- Who Can Create and Manage Action Categories or Templates? 18
- Viewing the Action Categories List 19
- Adding Custom Actions Using the CentraSite UI 19
- Adding Custom Actions Using APIs 23
- Viewing or Editing Action Categories or Templates 26
- Downloading Rules from System Action Templates 27
- Deleting Custom Action Categories and Templates 27
- Versioning a Custom Action Template 29
- Sample Custom Actions 30

Planning to Create Custom Actions

If you would like a policy to execute a task that is not provided by a built-in action, you can create a custom action to perform the work. For example, a custom action can consist of a Java class or a Groovy script 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 just like you would insert a built-in action.

Creating a custom action consists of the following high-level steps:

1. Create a custom *action category* with which to associate the action.
2. Create an *action template* to specify the scope of objects and events to which the action applies.
3. For an action template to be used in a design or change time policy, you specify the location of your custom *action rule*, which fires when the action executes.
4. Add *parameter templates* to define the parameters that serve as input to the action.

There are two ways you can add custom actions:

- You can use CentraSite Control to create action categories and action templates, as described in "Adding Custom Actions Using the CentraSite UI" on page 19. Then, you create action rules using Java programs or Groovy scripts, which is described in "Creating Action Rules" on page 23. You upload the rules into action templates when you create the action templates, using CentraSite Control.

  —OR—

- You can create action categories, action templates and action rules using Java programs or Groovy scripts, as described in "Adding Custom Actions Using APIs" on page 23. Then, you upload the rules scripts to action templates using JAXR-based calls in the action templates.

About 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 templates.

Predefined Action Categories Installed with CentraSite

By default, the policy actions that are installed with CentraSite are grouped into the following categories:

- Design-Time Category
- Change-Time Category

- Run-Time Category
- Global Category
- WS-I Category

For more information on the policy actions that CentraSite ships, see "Built-In Design/Change-Time Actions Reference" on page 37 and *Run-Time Governance with CentraSite*.

Custom Action Categories

If you would like 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.

About 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 is a manual action.

  Be aware that 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 fires an *action rule* when the action executes. You write an action rule as a Java class or a Groovy script. Programmatic actions are used only in design/change-time policies. CentraSite provides sample action rules. You upload action rules when you create a custom action template.

  You can upload action rules when you create a custom action template, as described in "Adding an Action Template to a Custom Action Category" on page 20.

For more information on the policy actions that CentraSite ships, see "Built-In Design/Change-Time Actions Reference" on page 37 and *Run-Time Governance with CentraSite*.

Supported Object Events

Although it is possible to create an action template whose scope encompasses any combination of object types and event types, be aware that not all combinations are enforceable. This is because certain types of events do not occur for certain types of objects.

To see the types of events that each object type supports, see the *CentraSite User's Guide*.

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 will never execute, because a PreStateChange event will never occur on a User object.

About 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 default 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.

Who Can Create and Manage Action Categories or 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

For more information about roles and permissions, see *Getting Started with CentraSite*.

Viewing the 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. 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-time or change-time or global or WS-I compliant action template and whether an action template type is manual or programmatic.

Adding Custom Actions Using the CentraSite UI

To add a custom action to CentraSite, perform the following high-level steps:

1. **Create a custom action category.** During this step, you specify the type of the category and details for the category.
2. **Add an action template to the custom action category.** During this step, you specify details for the action template, upload its associated action rule (if appropriate), and select the object and event types to which this template applies.
3. **Add a parameter template to the action template.** During this step, you add one or more parameter templates, and specify their input values.

Creating a Custom Action Category

Perform these steps to create a custom action category and save it to CentraSite.

To create a custom category

1. In CentraSite Control, go to **Policies > Action Templates**.

2. Click the **Add Action Category** button in 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 does not need to 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. Choose the type of template that the category will contain (for example, Design/Change-Time or Run-Time templates).

   c. Click **OK**.

Adding an Action Template to a Custom Action Category

Perform these steps to 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 yet exist, the **Add Action Category** dialog box is displayed, prompting you to create a custom action category. After you create a custom action category, the Add Action Template page is displayed, 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	Enter a name for the new action template. Follow these guidelines: ■ 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.
Implementation	For a programmatic action (a design-time or change-time action), specify whether the action's rule is a Groovy script or a Java class.
Uploaded File	For a programmatic action, click the **Browse** button and upload the action's rule file. ■ For a Java rule type, upload its Java program .zip file. ■ For a Groovy rule type, upload its .groovy script. For procedures on creating action rules, see "Creating Action Rules" on page 23. For procedures on uploading action rules, see "Uploading Action Rules to Action Templates" on page 25. **Note:** Alternatively, you can download the rules used by the system action templates, and modify them for use with the custom action template. For procedures, see "Downloading Rules from System Action Templates" on page 27.

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. For more information, see "Supported Object Events" on page 18.

6. Click **Save**.

   Upon saving the action template, CentraSite displays the Edit Action Template Detail page. You will use this page's **Parameter Templates** profile to add parameter templates for this action.

Adding a Parameter Template to the Action Template

To complete the action template, you must define its input parameters.

To add a parameter template to the action template

1. If you are beginning this procedure immediately after completing "Adding an Action Template to a Custom Action Category" on page 20, skip to step 4.
2. In CentraSite Control, go to **Policies > Action Templates**.
3. Select the action template for which you want to define parameter templates.
4. In the Edit Action Template Detail page, select the **Parameter Templates** profile.
5. Click the **Add Parameter Template** button.
6. Define the first parameter as follows:

In this field...	Do the following...
Name	Enter a name for the new parameter template.
Type	Select a data type.
Default Value	If you want to specify a default value, type a value in this field. If the selected data type is String, Number or URL, you can specify one or multiple default values. You can specify *multiple* possible default data values from which to choose as follows: a. Select the **Edit** icon to the right of the **Default Value** field. b. In the **Add Default Values** dialog box, type a value and click **Add**. Repeat for as many values as you need. c. Click **OK**. d. Then, in the **Default Value** field select from the drop-down list the value you want to use as the default value. **Note:** If you would rather fill in required values when this template is used in a policy, leave this field blank.

In this field...	Do the following...
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.

7. If you need to define additional parameters, click the **Add Parameter Template** button again and repeat the previous step.
8. Click **Save** and then **Close**.

   The parameter templates that you added appear under the **Parameter Templates** profile.

Adding Custom Actions Using APIs

To create a custom action programmatically, you perform the following high-level steps:

1. **Create a custom action category and template.**

   To do this, you create a Java class that uses the `com.centrasite.jaxr.CentraSiteLifeCycleManager` interface. To view the Javadoc for this interface, see the *CentraSite Java API Reference*.
2. **Create a Java class action rule or a Groovy script action rule.** For procedures, see "Creating Action Rules" on page 23.
3. **Upload the action rule to the action template.** For procedures, see "Uploading Action Rules to Action Templates" on page 25.

Creating Action Rules

Creating a Rule in a Java Class

Perform the following steps to create a rule in a Java class.

To create a rule in a Java class

1. 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 a .zip file that contains the following:

   - A folder named `lib`, which should contain a jar file with the action's executor class and the external libraries.
   - A folder named `META-INF`, which should 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 .zip file, as described in "Uploading Action Rules to Action Templates" on page 25.

   For example, the sample Java action rule provided in the your CentraSite installation has the following file structure:

   - *<CentraSite_Install_Dir>* \demos\Custom actions\Java\META-INF \assertions.properties

     This 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>
     ```

   - *<CentraSite_Install_Dir>* \demos\Custom actions\Java\src\com\softwareag \demo\actions\UniqueNameChecker.java

     This is the sample source file for the action executor. You can modify this file as needed, and compile it using the build file.

   - *<CentraSite_Install_Dir>* \demos\Custom actions\Java\build.xml

     This build file has the default target "zip", which will compile the Java file, build a jar out of it and pack it as a zip file.

   - *<CentraSite_Install_Dir>* \demos\Custom actions\Java\uniquenamechecker.zip

     This .zip file contains the following:

     - 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 the property file assertion.properties, which is the build file for the action.

   For more information about the sample Java action, see "Sample Custom Actions" on page 30.

Creating a Rule Using a Groovy Script

Perform the following steps to create a rule using Groovy script.

To create a rule using a Groovy script

- Upload the .groovy file as described in "Uploading Action Rules to Action Templates" on page 25.

Uploading Action Rules to Action Templates

Before you upload a Java action rule, you must first create a .zip file as described in "Creating Action Rules" on page 23.

If you want to upload a Groovy script rule, the .groovy file can contain the following variables. You cannot upload external libraries.

Variable	Description
`entity`	This represents the RegistryObject which is in the context. The following fields are available in this object: ■ Name ■ Description ■ State
`policyContext`	The PolicyContext in which the policy is running.
`assertion`	The IAssertionInstance that is in execution.
`result`	The result object that has the status and message. Set the result.successMessage or result.failureMessage. Based on the message, the status is inferred. If you fail to set a message, the status will be treated as a successful execution with an empty message.

To upload an action rule

There are two ways you can upload a rule's .zip file or .groovy file to a custom action template:

- If you are uploading the rule to a custom action template you created using the CentraSite Control, you upload the rule (either a Java .zip file or a .groovy file) when you create the action template, on the Add Action Template page. For procedures, see "Adding an Action Template to a Custom Action Category" on page 20.

  —OR—

- If you are uploading the rule to a custom action template that you created programmatically, you upload the rule (either a Java .zip file or a .groovy file) using a JAXR-based call in the action template.

Viewing or Editing Action Categories or Templates

You use the Edit Action Category or Edit Action Template page to examine and/or edit the properties of an action category or action template. When viewing or changing the properties of an action category or template, keep the following points in mind:

- You cannot edit or delete the predefined categories or action templates that are installed with CentraSite.
- You can change any property of an custom category or action template; however cannot modify the type.
- You can edit an category or action template only after deactivating all the policies that use it.
- You can rename an action category at any time.

Viewing or Editing an Action Category

You use the following procedure to view or edit the action category details.

To view or edit 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 edit.
3. Examine or modify the category's properties on the **Edit Action Category** dialog box as appropriate. For more information about these properties, see "Creating a Custom Action Category" on page 19.

Viewing or Editing an Action Template

You use the following procedure to view or edit the action template details.

To view or edit 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 edit.
3. Examine or modify the template's properties on the Edit Action Template page as appropriate. For more information about these properties, see "Adding an Action Template to a Custom Action Category" on page 20.

4. Select the **Scope** profile. View or edit 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. See "Supported Object Events" on page 18.

5. Select the **Parameter Templates** profile. View or edit the parameter fields as appropriate.
6. If you have edited a template's properties, click **Save**. Otherwise, click **Cancel**.

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. Choose the .zip file in the **Uploaded File** field, and then download the rule.

Structure of the Zip File

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>
```

Deleting 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 of 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 check box beside the parameter template name and click **Delete**.

   This temporarily revokes the selected parameter template from the action template.
5. Click **Save** to permanently remove the parameter template from the action template.

Deleting a Custom Action Template

Before you attempt to delete a custom action template, you must first delete all of the policies consuming it. Also, be aware 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 check box beside the action template name and click **Delete**.

Deleting a Custom Action Category

Before you attempt to delete a custom action category, you must first delete all of the policies consuming it.

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 check box beside the action category name and click **Delete**.

Versioning a Custom Action Template

If you need to modify a custom action template, you can create a new version of the existing template and make your changes to the new version. When you create a new version of a custom action template, CentraSite creates an identical copy of the existing template, and then you make your changes to the copy. (Note that the new version of the custom action template will get is own copy of the executable Groovy or Java file.)

Be aware 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 will 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 *will 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 will only allow you to create a new version of the action from version 3.0.
- Initially, the new version of the action will be 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 and the previous version. CentraSite uses this relationship to enforce rules related to versioned actions.
- You can only create new versions of *custom* actions that exist on your instance of CentraSite (i.e., 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 necessary and then save it.

   Tip: To make the new version of the custom action easy to distinguish from earlier versions, consider appending the version number to the name of the custom action. This will make the versions easier to tell apart when you view or edit the action list for a policy.

Sample Custom Actions

Your CentraSite installation contains two sample custom action rules. One rule is a Java rule, and the other is a Groovy script rule.

Sample Java Action: Enforce Unique Asset Names

Your CentraSite installation contains a sample Java action rule (which is contained in uniquenamechecker.zip) that you can use to create a custom action that ensures that the name and version combination of a newly-created asset is unique within the CentraSite catalog. If it is not unique, the action returns Failure, and the asset is not allowed to be created.

To create the custom action, you will use the CentraSite user interface 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. Choose **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. On the Add Action Template page, specify the following fields:

In this field...	Do the following...
Category	Select the custom action category you just created.

In this field...	Do the following...
Name	Enter the name `Enforce Unique Asset Names` for the new action template.
Description	*Optional*. Type a description for the new action template. For example: `Ensures that asset names are unique.`
Implementation	Select **Java**.
Uploaded File	Click the **Browse** button and upload the following rule file: *<CentraSite_Install_Dir>* \demos\Custom actions\Java \uniquenamechecker.zip This .zip file that contains the following: ■ 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. For more information about creating and uploading action rules, see "Creating Action Rules" on page 23.

6. In the **Scope** panel, specify the following fields:

In this field...	Do the following...
Object Types	Select **Service** as the type of object to which this action template applies.
Event Types	Select **PreCreate** 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:

   a. In CentraSite Control, go to **Policies > Design/Change Time**.

   b. Click **Add Policy**.

c. In the **Policy Information** panel, enter a name for the new policy, for example, Ensure Unique Asset Names Policy. A policy name can contain any character (including spaces).

d. 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 **PreCreate** 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 will be applied).

e. Click **Next**.

f. From the **Available Actions** list, choose the custom action **Enforce Unique Asset Names** action that you created.

g. Click **Finish** to save the new policy. The Design/Change-Time Policy Details page is displayed.

h. Activate the policy by choosing the **Change State** button and choosing the **Productive** state.

Sample Groovy Script Action: Service Attribute Checker

Your CentraSite installation contains a sample Groovy action rule (ServiceAttributeChecker.groovy) that you can use to create a custom action that checks for a particular value of a service attribute.

To create and test the custom action, you will use the CentraSite user interface to:

1. Create a custom action category
2. Create a custom action template, to which you upload the sample Groovy script rule
3. Create a Design/Change-Time policy and add the custom action template to it
4. Test the custom action "on demand" (manually) on the policy's detail page

To create and test the custom action Service Attribute Checker

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. Choose **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. On the Add Action Template page, specify the following fields:

In this field...	Do the following...
Category	Select the custom action category you just created.
Name	Enter the name `Service Attribute Checker` for the new action template.
Description	*Optional*. Type a description for the new action template. For example: `Validates asset attribute values.`
Implementation	Select **Groovy**.
Uploaded File	Click the **Browse** button and upload the following rule file: <*CentraSite_Install_Dir*> \demos\Custom actions \Groovy\ServiceAttributeChecker.groovy

In the **Scope** panel, specify the following fields and click **Save**:

In this field...	Do the following...
Object Types	Select **Service** as the type of object to which this action template applies.
Event Types	Select **OnTrigger** as the type of event to which this action template applies. This will enable you to test the action "on demand" (manually) in the **Actions** profile of the Design/Change Time Policy Detail page.

6. In the Edit Action Template Detail page, select the **Parameter Templates** profile in order to add the action's parameter templates and click the **Add Parameter Template** button.
7. Set the first parameter template for the action as follows:

In this field...	Do the following...
Name	Enter `Attribute Name`. This parameter will hold the attribute that you want to check (e.g., any CentraSite attribute, which you will assign after you add the action to a policy).
Type	Choose the data type **Attribute** for this parameter template.
Default Value	Leave blank.
Array	Leave blank.
Required	Select this check box.

8. Click the **Add Parameter Template** button again to set the second parameter template as follows:

In this field...	Do the following...
Name	Enter `Possible Attribute Value`. This parameter will hold the value of the attribute that you want to check.
Type	Select the data type **String** for this parameter template.
Default Value	Enter an attribute value that you want to check.
Array	Leave blank.
Required	Select this check box.

9. Click **Save** and then **Close**.

   The parameter template that you added appears in the **Parameter Templates** profile.

10. Create a policy and add the sample action to the policy as follows:

    a. In CentraSite Control, go to **Policies > Design/Change Time**.

    b. Click **Add Policy**.

    c. In the **Policy Information** panel, enter a name for the new policy, for example, `Service Attribute Checker Policy`. A policy name can contain any character (including spaces).

d. 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 **OnTrigger** 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 will be applied).

e. Click **Next**.

f. From the **Available Actions** list, choose the custom action **Service Attribute Checker** action that you created.

g. Click **Finish** to save the new (as yet incomplete) policy. The Design/Change-Time Policy Details page is displayed.

h. To configure the two parameters for the action, choose the action name on the **Actions** profile. For the **Attribute Name** parameter, choose an attribute. For the **Possible Attribute Value** parameter, select the default value that you defined for the **Possible Attribute Value** parameter in the action template.

i. Click **Save** and then **Close**. The policy detail page is displayed.

j. Activate the policy by choosing the **Change State** button and choosing the **Productive** state.

k. Select the **Actions** profile.

l. To test the action, click the **Run** button. The action checks all services in your organization and displays a pop-up. The pop-up should indicate Success in the Result column for the service that matches the value you specified in `Possible Attribute Value`. All other services will also be displayed, with Failure in the Result column.

11. After you have completed testing, you can use this custom action in other Design/Change-Time policies. To do this, go to its Scope panel, change the event type from **OnTrigger** to **PreCreate** or **PreUpdate**, and include the action in other Design/Change-Time policies whose event type scope is either PreCreate or PreUpdate.

2 Built-In Design/Change-Time Actions Reference

- Summary of Actions in the ARIS Category 38
- Summary of Actions in the Change-Time Category 38
- Summary of Actions in the Collector Category 40
- Summary of Actions in the Design-Time Category 41
- Summary of Actions in the Global Category 41
- Summary of Actions in the Handler Category 42
- Summary of Actions in the WS-I Category 43
- Built-In Actions for Design/Change-Time Policies 44

Summary of Actions in the ARIS Category

The following action templates are available in the ARIS category:

Action Template	Description
Notify ARIS Service	Notifies the ARIS APG service endpoint when: ■ A Process object in CentraSite is updated or deleted. ■ A Service object (native or virtual) in CentraSite is updated or deleted, or when a user changes the state of the service to a “completed” lifecycle state (e.g, the Productive state).

Summary of Actions in the Change-Time Category

The following action templates are available in the Change-Time category:

Action Template	Description
Change Activation State	Activates or deactivates a lifecycle model or a policy.
Change Deployment Status	Enables or disables the deployment of a virtual service.
Classify	Classifies an object by one or more taxonomy categories.
Delete RuntimeEvents and RuntimeMetrics	Deletes the logged events and metrics associated with a service.
Initiate Approval	Initiates an approval workflow.
Initiate Group-Dependent Approval	Initiates an approval workflow based on the group to which the requestor belongs.
Mark Pending on Runtime Policy Change	Marks a service as pending for redeployment on activation or deactivation of the applicable run-time policy.

Action Template	Description
Processing Steps Status	Enables or disables the **Processing Steps** profile for a virtual service.
Promote Asset	Promotes an asset to a new lifecycle stage (moving from one CentraSite instance to another CentraSite instance).
Register Consumer	Registers users and/or consumer applications as consumers of the requested asset.
Set Attribute Value	Assigns a value to a specified attribute in an organization, user or asset object.
Set Consumer Permission	Gives consumers instance-level permissions on the asset for which they have been registered.
Set State	Changes the lifecycle state of a lifecycle model, policy or asset.
UnClassify	Removes specified taxonomy categories from an object.
Validate Attribute Value	Validates the value of a specified attribute in an organization, user or asset against a list of allowed values.
Validate Classification	Checks whether an object is classified by a given taxonomy or taxonomy category.
Validate Lifecycle Model Activation	Checks whether a lifecycle model is ready to be activated.
Validate Policy Activation	Checks whether a policy is ready to be activated.
Validate Policy Deactivation	Verifies that a policy is not currently in-progress (i.e., undergoing execution) so that it can be successfully deactivated.
Validate State	Validates the current state of a lifecycle model, policy or asset against a given list of states.

Summary of Actions in the Collector Category

The following action templates are available in the Collector category:

Important: The actions in this category are used by the predefined collector policies that are installed with CentraSite. They are not intended to be used in user-defined policies. For information about predefined collector policies, see the *CentraSite User's Guide*.

Action Template	Description
BPEL Collector	Performs the collection process on BPEL Process objects
Default Collector	Performs the collection process for types that do not have a specified collector.
Lifecycle Model Collector	Performs the collection process on Lifecycle Models.
Policy Collector	Performs the collection process on Policy objects (both design/change-time policies and run-time policies)
REST Service Collector	Performs the collection process on REST Service objects.
Schema Collector	Performs the collection process on XML Schema objects
Virtual REST Service Collector	Performs the collection process on Virtual REST Service objects.
Virtual Service Collector	Performs the collection process on Virtual Service objects.
Virtual XML Service Collector	Performs the collection process on Virtual XML Service objects.
Webservice Collector	Performs the collection process on Service objects.
WS-Policy Collector	Performs the collection process on WS-Policy objects.

Action Template	Description
XML Service Collector	Performs the collection process on XML Service objects.
IS Service Interface Collector	Performs the collection process on IS Service Interface objects.

Summary of Actions in the Design-Time Category

The following action templates are available in the Design-Time category:

Action Template	Description
Validate Description	Validates the description of an object against a given pattern string.
Validate Name	Validates the name of an object against a given pattern string.
Validate Namespace	Checks that the target namespace attribute in a Service or XML Schema matches one of the valid namespaces in a given list.
Validate Service Binding	Checks that a Service supports the specified bindings.
Validate WSDL Size	Checks the size of the WSDL document associated with a Service to ensure that it falls within a specified range.
webMethods REST Publish	Creates a REST service in CentraSite from the published IS service interface object.

Summary of Actions in the Global Category

The following action templates are available in the Global category:

Action Template	Description
Call Web Service	Submits a given SOAP message to a specified Web service.
Enforce Unique Name	Ensures that the names of the Application Server type objects that are created in CentraSite are unique.
On Consumer Registration Request Send Email to Owner	Sends an email message to an object's owner when there is a consumer registration request for the object.
Publish to API-Portal	Publishes API metadata to API-Portal repository.
Send Email Notification	Sends an email message to a specified group of users.
Set Instance and Profile Permissions	Assigns instance-level permissions to an asset and to the asset's profiles.
Set Permissions	Sets instance-level permissions on an policy.
Set Profile Permissions	Assigns instance-level permissions to an asset's profiles.
Set View Permission for Service and Service Related Object to Everyone Group	Grants View permission to all users (including guests) on a given service.
Send Email Notification to Watchers	Sends an email notification to the watchers for an asset who are specific users asked to be notified for any modifications on that particular asset.
UnPublish from API-Portal	Revokes API metadata form API-Portal repository.

Summary of Actions in the Handler Category

The following action templates are available in the Handler category:

Important: The actions in this category are used by the predefined handler policies that are installed with CentraSite. They are not intended to be used in user-defined policies. For more information about the predefined handler policies, see the *CentraSite User's Guide*.

Action Template	Description
Asset Type Export Handler Action	Handler that CentraSite uses to export Type objects.
Default Delete Handler	Handler that CentraSite uses to delete instances of types that do not have a their own delete handlers.
Default Export Handler Action	Handler that CentraSite uses to export instances of types that do not have their own export handlers.
Default Move Handler	Handler that CentraSite uses to move instances of types that do not have their own move handlers (move to another user and/or to another organization).
Default User Move Handler	Handler that CentraSite uses to move users to other organizations.
Organization Export Handler Action	Handler that CentraSite uses to export organizations.
Reject Handler Action	Handler that prevents instances of a type from being deleted, exported, or moved, except as part of a composite object.
Taxonomy and Category Export Handler Action	Process that CentraSite uses to export taxonomies and their categories.
Virtual Service Export Handler Action	Process that CentraSite uses to export Virtual Service objects.

Summary of Actions in the WS-I Category

The WS-I category contains numerous actions from Basic Profile 1.1 and SSBP 1.0 that you can use to test a Web service (of type Service or Virtual Service) for compliance with Web Service Interoperability (WS-I) standards.

Important: A policy that contains WS-I actions must not contain any other type of action. If you need to execute other types of actions for the same event, you must place those actions in a separate policy.

Built-In Actions for Design/Change-Time Policies

Call Web Service

Submits a given SOAP message to a specified Web service. You can use this action to notify external systems, via a SOAP message, of changes that occur in the registry.

If the Web service returns a response, the response message is recorded to the policy log.

If the Web service produces a SOAP fault or the service cannot be successfully performed for other reasons (e.g., a network failure occurs), the policy action fails, and thus the policy itself fails. If the policy had been executed on a "pre" operation event (e.g., PreCreate, PreDelete), the requested operation is not executed.

Event Scope

PreCreate
PostCreate
PreUpdate
PostUpdate
PreDelete
PostDelete
PreStateChange
PostStateChange
OnTrigger

Object Scope

This action can be enforced on any object type that the policy engine supports.

Input Parameters

`Service Endpoint`	*String* The URL of the Web service that you want to call. Supported protocols are HTTP and HTTPS. Example: `http://myServer:53307/wsstack/myService` **Note:** If the Web service that you want to invoke is registered in CentraSite, you can use the **Browse** button to select its URL.

`HTTP Basic Auth Enabled`	*Boolean* Specifies whether the service is secured by Basic HTTP authentication. If you enable this option, you can optionally specify the user ID and password that CentraSite is to submit when it invokes the service in the following parameters. If you leave these parameters empty, CentraSite will submit the credentials belonging to the user who triggered this policy action.

`HTTP Basic Auth Username`	The user ID that you want CentraSite to submit for HTTP basic authentication (if you do not want CentraSite to submit the user ID of the user who triggered the policy).
`HTTP Basic Auth Password`	The password associated with the user ID specified in `HTTP Basic Auth Username`.

`SOAP Request Message` — *String* The SOAP message that CentraSite is to submit to the Web service. This message can include substitution tokens, if you want to insert run-time data into it. For available tokens, see the list of Substitution Tokens shown in the Send Email Notification action.

```
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <m:keylogger
        xmlns:m=" http://mycompany.example.org/key ">
          <serviceName>${entity.name}</serviceName>
          <assetType>${entity.type}</assetType>
          <key>${entity.attribute.Key}</key>
    </m:keylogger>
  </env:Body>
</env:Envelope>
```

`SOAP Action`	*String* The SOAP action that CentraSite will set in the message. If you do not set this parameter, CentraSite will set the SOAP action to the empty string.
`Connection Timeout (in milliseconds)`	*Number* The length of time in milliseconds that CentraSite will wait for a response from the remote machine. If the timeout limit is exceeded, the policy action fails.
`Content Type`	*String* The value that CentraSite is to assign to the Content-Type header in the SOAP request that it submits to the service.

Example:

```
application/soap+xml; charset=utf-8
```

If you do not specify `Content Type`, the value, `application/soap+xml`, is assigned to the SOAP request.

Change Activation State

Activates or deactivates a lifecycle model or a policy.

Event Scope

PostStateChange
OnTrigger

Object Scope

Lifecycle Model
Policy

Input Parameters

`Change Activation State To`	*String* The activation state to which you want to set the lifecycle model or policy as follows:	
	Active	Activates the policy or lifecycle model. This action will fail if it attempts to activate: ■ A policy whose parameters are not set. ■ A lifecycle model that does not have an associated object type. ■ A lifecycle model whose associated object type is already assigned to another lifecycle model. To prevent these types of failures from occurring, you should always execute the appropriate validation action before changing the activation state of a policy or lifecycle model. See the following: ■ "Validate Policy Activation" on page 91 ■ "Validate Lifecycle Model Activation" on page 89
	Inactive	Deactivates the policy or lifecycle model.

The following options are used to create policies that support the automatic deactivation of an older version of a policy or lifecycle model when a newer version is activated. In a lifecycle model for policies or lifecycle models, any state during which a policy or lifecycle is active must include a transition that places the policy or lifecycle model in one of the following activation states.

For example, the default lifecycle model for policies includes the Productive state. This is the only state in the model during which the policy is active. The Productive state includes a transition to the Retired state, which triggers a policy that switches the policy's activation state to "Superseded and Retired".

Because the Productive state includes this transition, CentraSite is able to automatically deactivate an old version of a policy when a new version is activated. It simply locates and executes the transition that places the policy in one of the following states. In the case of policies, this transition is the one to the Retired state, which puts the policy in the "Superseded and Retired" state of activation.

Superseded	Deactivates the policy and switches the policy's activation state to `Superseded` to indicate that the policy has been replaced by a newer version.
Retired	Deactivates the policy and switches the policy's activation state to `Retired` to indicate that the policy is no longer available for use.
Superseded and Retired	Deactivates the policy and switches the policy's activation state to `Superseded and Retired` to indicate that the policy has been replaced by a new version and is no longer available for use.

This action will fail if it attempts to deactivate a policy that is in-progress. To prevent this type of failure from occurring, you should always execute the "Validate Policy Deactivation" on page 92 action before using the Change Activation State action to deactivate a policy or lifecycle model.

Change Deployment Status

Enables or disables the deployment status of a virtual service. You use this action to specify whether the given virtual service is eligible or ineligible for deployment.

- When you enable the deployment status for a virtual service, you enable the controls on the **Deployment** profile. These controls enable authorized users to deploy, undeploy or redeploy the virtual service.

  Additionally, enabling the deployment status of a virtual service makes the virtual service eligible for automatic re-deployment when changes occur to its run-time policies.

- When you disable the deployment status for a virtual service, you disable the controls on the virtual service's **Deployment** profile (thus, preventing users from deploying, undeploying or redeploy the virtual service).

  When the deployment status for a virtual service is in the disabled state, the virtual service is not eligible for automatic re-deployment when changes occur to its run-time policies.

  Note: Disabling the deployment status of a virtual service *does not* undeploy the virtual service if it is already deployed. If the virtual service is currently deployed on a Mediator, it remains deployed there. However, administrators will not be able to undeploy or redeploy the virtual service from CentraSite Control until its deployment status is enabled.

To enable the deployment status of a virtual service, the following conditions must be satisfied:

- There must be at least one target defined in the registry.
- The Entry Protocol and Routing steps must be configured.

Typically, you use this action in combination with the "Processing Steps Status" on page 64 action, which enables and disables the **Processing Steps** profile for a virtual service. For example, when you enable the **Deployment** profile, you generally disable the **Processing Steps** profile and vice versa.

Event Scope

PostStateChange

Object Scope

Service
Virtual Service
Virtual REST Service
Virtual XML Service

Input Parameters

`Enable Deployment`	*Boolean* Specifies whether the virtual service is eligible for deployment (parameter set to "Yes") or ineligible for deployment (parameter set to "No").

Classify

Classifies the target object (i.e., the object on which the policy was triggered) by one or more taxonomy categories. You can assign the taxonomy categories to a classification attribute of the target object, or you can assign the taxonomy categories as normal classifications of the target object.

The classifications you assign using this action will appear on the asset's **Classification** tab. The classifications you assign will also appear for the selected classification attribute.

You can choose whether the classifications you specify with this action will be added to the object's existing classifications or whether they will replace the object's existing classifications. This choice is only available for multi-value classification attributes, i.e. classification attributes that can reference more than one taxonomy category. If a classification attribute is a single-value classification attribute, its existing value will be replaced by the new one.

Event Scope

PostCreate
PostStateChange
OnTrigger
OnConsumerRegistration

Object Scope

This action can be enforced on any object type that the policy engine supports.

Input Parameters

`Classify With Attribute`	*Object Array* This holds the parameters `Classification Attribute` and `Categories`.
`Classification Attribute`	*String (optional)* This specifies the name of the object's attribute to which the following classification categories apply. If you leave this parameter empty, the classification categories will be used as normal classifications of the target object.

`Categories`	*Taxonomy Node Array* The taxonomy nodes by which you want to classify the object.
`Overwrite`	*Boolean* If true, this specifies that you want to overwrite all existing classifications with the newly specified classifications. If false, the newly specified classifications are added to the existing classifications. **Note:** This option applies only to multi-value classification attributes. If a classification attribute is a single-value classification attribute, its existing value will be replaced by the new one, regardless of the setting of the Overwrite parameter.

Consumer WSDL Generator

Enables the **Consumer WSDL** option on the Specification profile of SOAP-based virtual services. For information about the Consumer WSDL option, see information about the Specification profile in *Run-Time Governance with CentraSite*.

Event Scope

PreCreate
PreUpdate
OnTrigger

Object Scope

Virtual Service

Input Parameters

None.

Default Move Handler

Performs standard actions when an object's owner or organization changes.

This action is included in the *Default Move Handler* policy that is installed with CentraSite. This is the default policy that executes when an object is moved to a new owner or organization.

Event Scope

OnMove

Object Scope

This action can be enforced on any object type that the policy engine supports.

Input Parameters

`Send Notification`	*Boolean* Specifies whether a notification should be sent to the object's new owner and previous owner. If this is set to `true`, a notification will be sent to the new owner and the previous owner. Also, a subscription to the object will be created automatically for the new owner and the previous owner. If the ownership changes again at a later time, the subscriptions of the old owners (i.e. users who owned the object before the new owner and the immediate previous owner) will not be automatically deleted, so the old owners will continue to receive notifications of ownership changes until they delete the subscription explicitly. If this is set to `false`, no notification will be sent to the new owner or the previous owner. However, a notification will be sent to any other user who has a subscription to be notified of an ownership change for the object. The default is `true`.

Delete RuntimeEvents and RuntimeMetrics

Deletes the events and metrics that have been logged for a service.

This action is included in the *Delete RuntimeEvents and RuntimeMetrics of Service* policy that is installed with CentraSite. This policy executes when a service is deleted. The policy ensures that the metrics and events associated with a service are removed from the run-time logs when a service is deleted.

Event Scope

PreDelete

Object Scope

Service
Virtual Service
REST Service
Virtual REST Service
XML Service
Virtual XML Service
CEP Event Type

Input Parameters

`Delete Runtime Events`	*Boolean* Specifies whether the events that have been logged for a service are to be deleted.
`Delete Runtime Metrics`	*Boolean* Specifies whether the runtime metrics that have been logged for a service are to be deleted.

Enforce Unique Name

Ensures that the names of objects that are created in CentraSite are unique.

This action is included in the *Enforce Unique Name* policy that is installed with CentraSite. For information about this policy, see the information about using CentraSite with ARIS in the *CentraSite Administrator's Guide*.

Event Scope

PreCreate
PreUpdate
PreStateChange
OnTrigger

Object Scope

This action can be enforced on any object type that the policy engine supports.

Input Parameters

`Enforce Across Organizations`	*Boolean* If this parameter is set to True, then the unique name requirement for objects is enforced in all organizations defined in CentraSite.
`Allow Different Versions`	*Boolean* If this parameter is set to True, then different versions of an object can exist in CentraSite with the same name.

Initiate Approval

Initiates an approval workflow.

When this action is executed, CentraSite initiates the approval process. CentraSite will not process any subsequent actions in the policy or execute the requested operation until the approvals specified by the Initiate Approval action are received.

Caution: When you use this action on the PreStateChange event, only certain kinds of actions can be executed *after* this action in an approval policy. Some actions, if they occur after this action, will cause the policy to fail.

Note: To use the email options provided by this action, CentraSite must have a connection to an SMTP email server. For instructions on how to configure CentraSite's connection to an email server, see the *CentraSite Administrator's Guide*.

If You Migrate this Action from a Pre-8.2 Release

If you have a policy that contains this action and the policy was created prior to version 8.2, that policy will continue to exhibit the old email-notification behavior (i.e., it will continue to send the earlier version's standard email message to approvers). If you want to use the email-notification enhancements that were introduced in version 8.2, simply edit the policy and enable the email parameters in the Initiate Approval action.

Event Scope

PreStateChange
OnConsumerRegistration

Object Scope

This action can be enforced on any object type that the policy engine supports.

Input Parameters

`User`	*String* The user name that will be used together with the Password parameter as authentication credentials for performing a lifecycle model state change on a service asset. The credentials are stored in the approval request and passed to the web service for completing the approval. The user specified must have the permissions required to perform the state change. This parameter is only visible to users with the CentraSite Administrator role.
`Password`	*String* The password that will be used together with the User parameter as authentication credentials. This parameter is only visible to users with the CentraSite Administrator role.
`Approval Flow Name`	*String* The name to be given to the approval workflow that this action initiates. This name serves to identify the workflow in the Approval History log and in the approver's inbox.

	An approval flow name can contain any combination of characters, including a space. You can also include substitution tokens in the name to incorporate data from the target object on which the policy is acting. For a list of the allowed tokens, see the list of Substitution Tokens shown in the Send Email Notification action.
`Approver Group`	*String Array* The user group (or groups) that identifies the set of users who are authorized to approve the requested operation. **Note:** If the user groups specified in `Approver Group` are empty at enforcement time, the user's request is auto-approved.
`Approval is Needed From`	*String* The manner in which the approval is to be processed:

Value	Description
`AnyOne`	*Default* The request can be approved or rejected by any single user in `Approver Group`. In this mode, only one user from the set of authorized approvers is required to approve or reject the request.
`EveryOne`	The request must be approved by all users specified in `Approver Group`. (It does not matter in which order the approvals are issued.) A single rejection will cause the request to be rejected.

`Reject State`	The lifecycle state that is to be assigned to the object if the approval request is rejected. If this parameter is not specified, the object's lifecycle state does not change when a rejection occurs. The lifecycle model must define a valid transition from the state that the target object is in at the time it is submitted for approval to the state specified in `Reject State`. Otherwise, the target object's state will not be switched when a rejection occurs.
`Send Pending Approval Email`	*Boolean* Specifies whether CentraSite is to send an email message to specified users and/or groups when the request

is initially submitted for approval. If you enable this option, you must set the following parameters to specify the text of the message and to whom it is to be sent.

> **Note:** If the request is auto-approved, this message is not sent.
>
> **Note:** CentraSite automatically sends the email message to the approvers in addition to the users and/or groups that you specify below.

`Users`	*Array of Users* Users who are to receive the email. **Note:** You can specify the recipients of the email using the `Users` parameter, the `Groups` parameter, or both.
`Groups`	*Array of Groups* Groups whose users are to receive the email. **Note:** CentraSite will only send the email to those users in the group whose CentraSite user account includes an email address.
`Subject`	*String* The text that you want to appear in the subject line of the email. This text can include substitution tokens to insert run-time data into the subject line. For available tokens, see the list of Substitution Tokens shown in the Send Email Notification action.
`Use Email Template`	*Email Template* Specifies the template that is to be used to generate the body of the email message. **Note:** You can use the predefined template, PendingNotification.html, for pending-approval notifications if you do not want to create an email template of your own. **Note:** If you use an email template to generate the body of the message, you cannot specify the body of the message using the `Custom Message` parameter. (In

		other words, you specify the body of the message using either the `Use Email Template`*or* the `Custom Message` parameter.)
	`Custom Message`	*TextArea* The text of the email message. This text can include substitution tokens to insert run-time data into the message. For available tokens, see the list of Substitution Tokens shown in the Send Email Notification action. **Note**If you use the `Custom Message` parameter to specify the body of the email message, you cannot generate the body of the message using an email template. (In other words, you specify the body of the message using either the `Custom Message`*or* the `Use Email Template` parameter.)
	`Format`	*String* Specifies whether the message in the `Custom Message` parameter is formatted as HTML or plain text. For more information about using this option, see the *CentraSite User's Guide*.
	`Include owner in notification`	*Boolean* When the parameter is enabled, CentraSite sends the email to the owner of the object (on which the policy is acting) in addition to the other recipients.
`Send Approval Email`	*Boolean* Specifies whether CentraSite is to send an email message to specified users and/or groups when the request is approved. If you enable this option, you must set the following parameters to specify the text of the message and to whom it is to be sent. **Note:** CentraSite automatically sends the email message to the user who submitted the approval request in addition to the users and/or groups that you specify below. **Note:** When the EveryOne option is specified in the `Approval is Needed From` parameter, CentraSite sends this email only after all approvers have approved the request.	

	`Users`	See description of `Users` parameter.
	`Groups`	See description of `Groups` parameter.
	`Subject`	See description of `Subject` parameter.
	`Use Email Template`	See description of `Use Email Template` parameter. **Note:** You can use the predefined template, ApprovalNotification.html, for approval notifications if you do not want to create an email template of your own.
	`Custom Message`	See description of `Custom Message` parameter.
	`Format`	See description of `Format` parameter.
	`Include owner in notification`	See description of `Include owner in notification` parameter.
`Send Rejection Email`	*Boolean* Specifies whether CentraSite is to send an email message to specified users and/or groups when the request is rejected. If you enable this option, you must set the following parameters to specify the text of the message and to whom it is to be sent. **Note:** CentraSite automatically sends the email message to the approvers (except for the approver who rejected the request) and to the user who submitted the approval request in addition to the users and/or groups that you specify below.	
	`Users`	See description of `Users` parameter.
	`Groups`	See description of `Groups` parameter.
	`Subject`	See description of `Subject` parameter.
	`Use Email Template`	See description of `Use Email Template` parameter.

	Note: You can use the predefined template, RejectApprovalNotification.html, for rejection notifications if you do not want to create an email template of your own.
`Custom Message`	See description of `Custom Message` parameter.
`Format`	See description of `Format` parameter.
`Include owner in notification`	See description of `Include owner in notification` parameter.

Initiate Group-Dependent Approval

Initiates an approval workflow based on the group to which the requestor belongs. If the requestor does not belong to any of the groups specified in the Triggering Groups array, approval is waived and the action is considered to be completed successfully.

Caution: When you use this action on the PreStateChange event, only certain kinds of actions can be executed *after* this action in an approval policy. Some actions, if they occur after this action, will cause the policy to fail.

Note: To use the email options provided by this action, CentraSite must have a connection to an SMTP email server. For instructions on how to configure CentraSite‘s connection to an email server, see *CentraSite Administrator’s Guide*.

If You Migrate this Action from a Pre-8.2 Release

If you have a policy that contains this action and the policy was created prior to version 8.2, that policy will continue to exhibit the old email-notification behavior (i.e., it will continue to send the earlier version's standard email message to approvers). If you want to use the email-notification enhancements that were introduced in version 8.2, simply edit the policy and enable the email parameters in the Approval action.

Event Scope

PreStateChange
OnConsumerRegistration

Object Scope

This action can be enforced on any object type that the policy engine supports.

Input Parameters

`User`	*String* The user name that will be used together with the Password parameter as authentication credentials for performing a lifecycle model state change on a service asset. The credentials are stored in the approval request and passed to the web service for completing the approval. The user specified must have the permissions required to perform the state change. This parameter is only visible to users with the CentraSite Administrator role.
`Password`	*String* The password that will be used together with the User parameter as authentication credentials. This parameter is only visible to users with the CentraSiteAdministrator role.
`Approval`	*Object Array* The list of groups whose membership will determine whether the request requires approval, and if so, to which group of approvers the request is to be routed. Each object in the `Approval` array must contain the following information:

Parameter	Description
`Triggering Groups`	*String Array* The user group (or groups) that identifies the users whose requests must be approved.
`Approval Flow Name`	*String* The name to be given to the approval workflow that this action initiates. This name serves to identify the workflow when activity relating to it appears in the Approval History log or an approver's inbox. An Approval Flow Name can contain any combination of characters, including a space. You can also include substitution tokens in the name to incorporate data from the target object on which the policy is acting. For a list of the allowed tokens, see the list of Substitution Tokens shown in the Send Email Notification action.

`Approver Group`	*String Array* The user group (or groups) that identifies the set of users who are authorized to approve the requested operation. **Note:** If the user groups specified in `Approver Group` are empty at enforcement time, the user's request is auto-approved.
`Approval is needed from`	*String* The manner in which the approval is to be processed as follows:

Value	Description
AnyOne	*Default* The request can be approved or rejected by any single user in `Approver Group`. In this mode, only one user from the set of authorized approvers is required to approve or reject the request.
EveryOne	The request must be approved by all users specified in `Approver Group`. (It does not matter in which order the approvals are issued.) A single rejection will cause the request to be rejected.

`Reject State`

The lifecycle state that is to be assigned to the object if the approval request is rejected. If this parameter is not specified, the object's lifecycle state does not change when a rejection occurs.

The lifecycle model must define a valid transition from the state that the target object is in at the time it is submitted for approval to the state specified in `Reject State`. Otherwise, the target object's state will not be switched when a rejection occurs.

Mark Pending on Runtime Policy Change

Marks the deployed virtual services or consumer applications that are within the scope of run-time policy as pending for redeployment on activation or deactivation of the policy. After the policy is activated, the virtual services and consumer applications are automatically redeployed.

This action is included in the *Mark Pending-For-Redeployment On RuntimePolicy Change* policy that is installed with CentraSite. This policy executes when a run-time policy switches to the Productive state (which activates the policy) or Suspended state (which deactivates the policy).

If you customize the lifecycle model that CentraSite provides for policies and you add additional states to the model, you must execute this action during any transition that changes the activation state of a policy.

Event Scope

PreStateChange

Object Scope

Policy

Input Parameters

None.

Notify ARIS Service

Notifies the ARIS APG Service endpoint with the SOAP request message provided in this action. The APG Service endpoint is picked up from the associated ARIS Application Server.

You can use this action in the following policies:

- Notify ARIS on Process Changes
- Notify ARIS on Service Changes
- Notify ARIS on Service Completion
- Notify ARIS on Service Deletion

For information about using CentraSite with ARIS in the *CentraSite Administrator's Guide*.

Event Scope

PostUpdate
PostDelete
PostStateChange
OnTrigger

Object Scope

Process
Service

Input Parameters

`HTTP Basic Auth Enabled`	*Boolean* Specifies whether the service is secured by Basic HTTP authentication. If you enable this option, you can optionally specify the user ID and password that CentraSite is to submit when it invokes the service in the following parameters. If you leave these parameters empty, CentraSite will submit the credentials belonging to the user who triggered this policy action.

`HTTP Basic Auth Username`	The user ID that you want CentraSite to submit for HTTP basic authentication (if you do not want CentraSite to submit the user ID of the user who triggered the policy).
`HTTP Basic Auth Password`	The password associated with the user ID specified in `HTTP Basic Auth Username`.

`SOAP Request Message`	*String* The SOAP message that CentraSite is to submit to the ARIS service. This message can include substitution tokens, if you want to insert run-time data into it. For available tokens, see the list of Substitution Tokens shown in the Send Email Notification action.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://www.idsscheer.com/age/webMethods/">
  <soapenv:Header/>
  <soapenv:Body>
    <web:UpdateServiceRequest>
      <dbname>${context.ARIS_DB_CONTEXT}</dbname>
      <language>${user.locale}</language>
      <serviceDetail>
        <guid>${entity.key}</guid>
        <name>${entity.name}</name>
        <url>${entity.URL}</url>
        <lifeCycleState>${entity.state}</lifeCycleState>
        <owner>${entity.owner}</owner>
        <description>${entity.description}</description>
        <organization>${entity.organization}</organization>
        <version>${entity.version}</version>
        ${entity.attribute.Operations}
      </serviceDetail>
    </web:UpdateServiceRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

`SOAP Action`	*String* The SOAP action that CentraSite will set in the message. If you do not set this parameter, CentraSite will set the SOAP action to an empty string.
`Connection Timeout (in milliseconds)`	*Number* The length of time (in milliseconds) that CentraSite will wait for a response from the remote machine. If the timeout limit is exceeded, the policy action fails.
`Content Type`	*String* The value that CentraSite is to assign to the Content-Type header in the SOAP request that it submits to the service. Example: `application/soap+xml; charset=utf-8` If you do not specify `Content Type`, the value `application/soap+xml` is assigned to the SOAP request.

On Consumer Registration Request Send Email to Owner

This action template allows an email to be sent to the owner of an object if there is a consumer registration request for the object.

Event Scope

PreCreate

Object Scope

Consumer Registration Request

Input Parameters

`Custom Message`	*TextArea* The text of the email message. This text can include substitution tokens to insert run-time data into the message. For available tokens, see the list of Substitution Tokens shown in the Send Email Notification action. **Note:** If you use the `Custom Message` parameter to specify the body of the email message, you cannot generate the body of the message using an email template. (In other words, you specify the body of the message using either the `Custom Message`*or* the `Use Email Template` parameter.)
`Subject`	*String* The text that you want to appear in the subject line of the email. This text can include substitution tokens to insert run-time data into the subject line. For available tokens, see the list of Substitution Tokens shown in the Send Email Notification action.

`Format`	*String* Specifies whether the message in the `Custom Message` parameter is formatted as HTML or plain text.
`Use Email Template`	*Email Template* Specifies the template that is to be used to generate the body of the email message. This text can include substitution tokens to insert run-time data into the subject line. For available tokens, see the list of Substitution Tokens shown in the Send Email Notification action. **Note:** You can use the predefined template, PendingNotification.html, for pending-approval notifications if you do not want to create an email template of your own. **Note:** If you use an email template to generate the body of the message, you cannot specify the body of the message using the `Custom Message` parameter. (In other words, you specify the body of the message using either the `Use Email Template`*or* the `Custom Message` parameter.)

Processing Steps Status

Enables or disables the **Processing Steps** profile for a virtual service.

- When you enable the processing steps status for a virtual service, you enable the controls on the **Processing Steps** profile for that virtual service. These controls enable authorized users to modify the processing steps for the virtual service.
- When you disable the processing steps status for a virtual service, you disable the controls on the **Processing Steps** profile. While this profile is disabled, users cannot make changes to the virtual service's processing steps.

Typically, you use this action in combination with the "Change Deployment Status" on page 48 action, which enables and disables the **Deployment** profile for a virtual service. For example, when you enable the **Processing Steps** profile for a virtual service, you generally disable the **Deployment** profile and vice versa.

Event Scope

PostStateChange

Object Scope

Service
Virtual Service
Virtual REST Service
Virtual XML Service

Input Parameters

`Enable Processing Steps`	*Boolean* Specifies whether the Processing Steps profile for a virtual service is enabled (parameter set to "Yes") or disabled (parameter set to "No").

Promote Asset

This policy action allows you to promote an asset instance to a different CentraSite stage. The action can be executed on a lifecycle pre-state change, post-state change or on an OnTrigger event. The configurations cover the following options:

- **Specify a stage to promote to**

  This can be either the name of a lifecycle stage or the URL of the target registry.

- **Specify optional user credentials for the target stage**

  The credentials specify a user name and password of a user defined on the target registry. This user should have the required permissions to create the asset on the target registry.

- **Include referenced objects in the promotion set**

  Assets that are referenced by the asset being promoted can be included in the promotion process.

- **Keep the asset owner unchanged**

  You can specify that the owner of the asset in the source registry will also be the owner in the target registry. If this user does not exist in the target registry, the owner will be the user specified in the optional user credentials described above.

  This user should be able to create assets in the target organization, which can be any of the following, depending on the input parameters you specify:

  - The organization mentioned in the `Target Organization` parameter.
  - The organization to which the user in the target registry belongs.
  - The organization to which the triggering user or the user in the `Username` parameter belongs.

- **Replace existing registry objects in the target stage**

  If an asset already exists on the target stage, it may be replaced by the asset being promoted.

- **Specify a target organization name**

  When the asset is promoted, it will belong to the organization specified.

- **Keep the lifecycle state**

You can specify a lifecycle state for the promoted asset on the target registry. If you do not specify a state, the promoted asset will be placed in the initial state of the lifecycle model on the target registry.

Event Scope

PreStateChange
PostStateChange
OnTrigger

Object Scope

Asset

Input Parameters

The following table lists the input parameters for the policy action.

`Target Stage`	*String* The name of the target stage to which the asset will be promoted. This assumes that you have already defined the stage, as described in the section about lifecycle management in the *CentraSite Administrator's Guide*. If a value is specified for the parameter `Target Stage URL`, the value of `Target Stage` is used instead of the value of the parameter `Target Stage URL`. At least one of the parameters `Target Stage` or `Target Stage URL` must be specified, i.e. they cannot both be empty.
`Target Stage URL`	*String* The URL of the target CentraSite registry. If a value is specified for the parameter `Target Stage URL`, the value of `Target Stage` is used instead of the value of the parameter `Target Stage URL`. At least one of the parameters `Target Stage` or `Target Stage URL` must be specified, i.e. they cannot both be empty.
`Username`	*String (optional)* The user name and password are used as authentication credentials for the target stage. The assets will be created in the target by this user. If the user name and password are not supplied, the user name and password of the triggering user on the source stage will be used. If this user is not defined on the target stage, the promotion will fail.
`Password`	*String (optional)* The user name and password are used as authentication credentials for the target stage. The assets will be created in the target by this user.

	If the user name and password are not supplied, the user name and password of the triggering user on the source stage will be used. If this user is not defined on the target stage, the promotion will fail.
`Include Referenced Assets`	*Boolean (optional)* Specifies whether the referenced assets (referenced via associations) of the applied asset will be included for the promotion. A value of `yes` means that the references assets will also be promoted. A value of `no` means that only the specified asset will be promoted. The default value is `yes`.
`Keep Owner`	*Boolean (optional)* Specifies if the current owner will also be the owner in the target registry. This can only happen if the owner also exists as a user on the target registry and has the permissions required to create assets. A value of `yes` means that the asset owner on the target stage will be the same owner as on the source stage. A value of `no` means that the owner will be the specified user from the User Name parameter. The default value is `no`.
`Replace Existing Assets`	*Boolean (optional)* Specifies if an asset that already exists on the target stage may be replaced by the asset being promoted. A value of `yes` means that an asset on the target stage can be replaced. A value of `no` means that an existing asset on the target stage cannot be replaced. The default value is `no`.
`Keep Lifecycle State`	*Boolean (optional)* Specifies if the promoted asset should keep the lifecycle state that it has on the source stage. This can only happen if the lifecycle model used on the source stage is also defined and active on the target stage. A value of `yes` means that an asset on the target stage will have the same state as on the source stage. A value of `no` means that the promoted asset will be set to a lifecycle state according to the combinations as shown in the table below. The default value is `no`.

`Target Organization`	*String (optional)* Specifies the owning organization of the asset on the target stage. This can only happen if the specified organization exists on the target.

As noted in the table, some of the promotion operations are only possible if the target stage contains users, organizations and lifecycle models that are compatible with those defined on the source stage. The possible combinations are listed in the following tables.

Note: During the promotion process, CentraSite copies the metadata of an asset from the source instance to the target instance. However, if the action is to be executed during a prestatechange event, the changes due to the other actions in the source instance are not reflected in the target instance. You will need to explicitly update the asset if you want that change reflected in the target instance, too.

Important: Before you activate a policy that includes the Promote Asset action, ensure that the target's specified target stage URL or target stage is active and the user credentials of target registry are valid. To check this, click the **Check Connection** button. If the connection is not active and valid, activate the target specified in *Target Stage* or *Target Stage URL* and modify the user credentials as required.

Target Organization and Target Owner

When the asset is promoted to the target registry, it will belong to a specific organization and will be owned by a specific user. The organization and owner on the target registry are not necessarily the same organization and owner as on the source registry.

The owner on the target registry can be one of the following:

- the same owner as on the source registry (called "User A" in the following description)
- the user specified in the `Username` parameter (called "User B" in the following description)
- the triggering user, i.e. the user who activates the asset promotion (called "User C" in the following description)

The organization on the target registry can be one of the following:

- the organization specified in the `Target Organization` parameter (called "Organization P" in the following description)
- the organization of the user supplied in the `Username` parameter (called "Organization Q" in the following description)
- the organization of the triggering user (called "Organization R" in the following description)

Target Owner:

This user will be the owner ...	... under these circumstances
User A	If `Keep Owner` is specified, and User A has permission to create assets in Organization P or Q or R.
User B	If User A does not meet the requirements described in the previous row, and User B is defined.
User C	If User B not meet the requirements described in the previous row.

Target Organization

This organization will be the owning organization ...	... under these circumstances
Organization P	If `Target Organization` is specified and the target owner defined in the above table has permission to create assets in this organization.
Organization Q	If Organization P does not meet the requirements described in the previous row, and User B is defined.
Organization R	If Organization Q does not meet the requirements described in the previous row.

CentraSite attempts to create the asset on the target registry using the resulting combination of target owner and target organization. If the given user does not have permission to create assets in the given organization, the promotion will fail.

Keep Lifecycle State

Keep LCM State	Availability of the same LCM in the target stage	Does target have its own LCM	Result state of the promoted asset
yes	yes	n/a	Same state as in the source.

Keep LCM State	Availability of the same LCM in the target stage	Does target have its own LCM	Result state of the promoted asset
yes	no	yes	Initial state of the LCM in the target.
yes	no	no	No state assigned.
no	n/a	yes	Initial state of the LCM in the target.
no	n/a	no	No state assigned.

Publish to API-Portal

Enables you to publish API metadata to an API-Portal, thereby creating or updating the API information in the API-Portal repository.

Event Scope

- PreStateChange
- PostStateChange
- OnTrigger

Object Scope

- Service
- XML Service
- REST Service
- Virtual Service
- Virtual XML Service
- Virtual REST Service

Input Parameters

`API-Portal`	*Optional. String. Array.* The name of the API-Portal to which the API would be published. This assumes that you have already registered the API-Portal in CentraSite, as described in *Working with the CentraSite Business UI*. **Note:** However, if this action is to be executed in a different event other than OnTrigger, for example,

	prestatechange or poststagechange, that is not provided by default, you *must* specify a value for this field.
`Endpoint Category`	*Optional. String. Array.* The names of specific taxonomy categories by which the endpoints of the API are classified.
`REST Service Attributes`	*Optional. String. Array.* A metadata bundle can be supplied with additional information of a RESTful API and published to an API-Portal. You use this field to specify additional attributes of the REST API to be published to API-Portal.
`SOAP Service Attributes`	*Optional. String. Array.* A metadata bundle can be supplied with additional information of a SOAP-based API and published to an API-Portal. You use this field to specify additional attributes of the SOAP API to be published to API-Portal.

Register Consumer

Registers users, groups and/or consumer applications (as specified by the requestor) as consumers of an asset. This action creates a `consumed-by` relationship between the asset and the specified consumers. Once established, this relationship is visible in the asset's **Consumers** profile and also on the asset's Impact Analysis page.

The following actions are typically used in conjunction with the Register Consumer action.

- The approval actions (Initiate Approval or Initiate Group-Dependent Approval) are generally used to obtain necessary approvals prior to executing the Register Consumer action.
- The Set Consumer Permission action is typically executed after the Register Consumer action to give the specified consumers access to the requested asset.

Event Scope

OnConsumerRegistration

Object Scope

Asset (any type)

Input Parameters

None.

Send Email Notification

Sends an email message to specified users and/or groups.

Note: To use this action, CentraSite must have a connection to an SMTP email server. For instructions on how to configure the email server, see the *CentraSite Administrator's Guide*.

Note: During an iteration of the policy, if the connection to a SMTP email server fails, this policy action returns a failure code. CentraSite writes the failure message to the policy log; however performs the next action in the policy (if one exists).

Event Scope

PreCreate
PostCreate
PreUpdate
PostUpdate
PreDelete
PostDelete
PreStateChange
PostStateChange
OnConsumerRegistration
OnTrigger

Object Scope

This action can be enforced on any object type that the policy engine supports.

Input Parameters

`Users`	*Array of Users* Users who are to receive the email. **Note:** You can specify the recipients of the email using the `Users` parameter, the `Groups` parameter, or both.
`Groups`	*Array of Groups* Groups whose users are to receive the email. **Note:** CentraSite will only send the email to those users in the group whose CentraSite user account includes an email address.
`Subject`	*String* The text that you want to appear in the email's subject line. This text can include substitution tokens to insert run-time data into the subject line.

`Use Email Template`	*Email Template* Specifies the template that is to be used to generate the body of the email message. **Note:** You can use the predefined template, ChangeNotification.html, as your email template if you do not want to create an email template of your own. **Note:** If you use an email template to generate the body of the message, you cannot specify the body of the message using the `Custom Message` parameter. (In other words, you specify the body of the message using either the `Use Email Template` *or* the `Custom Message` parameter.)
`Custom Message`	*TextArea* The text of the email message. This text can include substitution tokens to insert run-time data into the message. **Note:** If you use the `Custom Message` parameter to specify the body of the email message, you cannot generate the body of the message using an email template. (In other words, you specify the body of the message using either the `Custom Message`*or* the `Use Email Template` parameter.)
`Format`	*String* Specifies whether the custom mail message is formatted as HTML or plain text.
`Include owner in notification`	*Boolean* When enabled, this parameter sends the email notification to the owner of the object on which the policy is acting in addition to the users specified by the `Users` and `Groups` parameters.

Substitution Tokens

The following list describes substitution tokens that you can use to incorporate data from the run-time instance of a policy into the email. For example, you can use tokens to return information about the object on which the policy is acting, identify the user who triggered the policy, and/or indicate what type of event caused the policy to fire.

Be aware that some tokens are only meaningful for certain types of objects. User objects, for example, do not have a Description attribute, so the ${entity.description} token has no meaning for a User object. If you use a substitution token that is not supported by the policy's target object, CentraSite simply replaces the substitution token with a space at enforcement time.

If the target object includes the requested attribute, but the attribute itself has no value, CentraSite also replaces the substitution token with a space in the email message. If the requested attribute contains an array of values, CentraSite inserts the values into the email as a comma-separated list.

This token...	Inserts the following information into the parameter value at execution time...
`${api.usage}`	This token will be replaced with the value specified in the `API Usage` attribute of API.
`${entity.approver}`	The name of the user who approved or rejected the approval request. **Note:** This token is only meaningful in email messages that are issued by the "Initiate Approval" on page 52 or "Initiate Group-dependent Approval" on page 58 actions. If it is used in a context where there is no approver or approval request, the token is simply replaced with a space.
`${entity.approvercomments}`	The comment provided by the approver when he or she approved or rejected the approval request. **Note:** This token is only meaningful in email messages that are issued by the "Initiate Approval" on page 52 or "Initiate Group-dependent Approval" on page 58 actions. If it is used in a context where there is no approval request, the token is simply replaced with a space.
`${entity.attribute.`*`attributeName`*`}`	The value of the attribute specified in *attributeName* . You can use this token with all attribute types (including computed types) except Classification, File, and Relationship types. **Important:** You must specify the attribute's schema name in *attributeName*, not its display name. For information about an attribute's schema name, see the *CentraSite Administrator's Guide*.
`${entity.BUIAssetURL}`	A link to the URL of the asset details page in CentraSite Business UI.

This token...	Inserts the following information into the parameter value at execution time...
`${entity.BUIBaseURL}`	A link to the URL of the CentraSite Business UI.
`${entity.description}`	The object's description. **Note:**Users do not have a Description attribute.
`${user.displayname}`	The display name of the user who triggered the policy.
`${entity.key}`	The object's key (i.e., the UUID that uniquely identifies the object within the registry).
`${entity.name}`	The object's name (in the user's locale).
`${entity.owner}`	The name of the user who owns the object against which the policy is acting.
`${entity.type}`	The type of object against which the policy acting.
`${entity.state}`	The state of the object against which the policy is acting. If the object is an Asset, Policy or Lifecycle Model, this action inserts the object's current lifecycle state. For all other object types, this token is ignored.
`${entity.URL}`	The URL for the object on which the policy is acting. (This is the URL that opens the object in CentraSite Control.)
`${entity.version}`	The object's user-assigned version identifier.
`${event.type}`	The type of event that triggered the policy.
`${from.state}`	The state from which the object is being switched (if the policy is executing on a PreStateChange or PostStateChange event.)

This token...	Inserts the following information into the parameter value at execution time...
`${target.state}`	The state to which the object is being switched (if the policy is executing on a PreStateChange or PostStateChange event).
`${user.locale}`	The locale of the user who triggered the policy.
`${user.name}`	The name of the user who triggered the policy.
`${user.organization}`	The name of the organization to which the user who triggered the policy belongs.

Example

```
User ${entity.owner} has added the following asset to the catalog:
Name: ${entity.name} Description: ${entity.description}
```

Set Attribute Value

Assigns a value to a specified attribute in an organization, user or asset.

Event Scope

Post-State Change
OnTrigger
OnConsumerRegistration
Pre-Create
Post-Create

Object Scope

Organization
User
Asset (any type)

Input Parameters

`Attribute Name`	*String/Non-String* The name of the attribute that you want to set. **Note:** `Attribute Name` must be a non-arrayed String/Non-String attribute.

Set Consumer Permission

Assigns permission settings to the users and/or groups who are identified by a consumer-registration request.

The behavior of this action with respect to specific asset profiles depends on the policy's object scope.

- If you use this action in a policy that applies to multiple asset types, you can set only the asset's top-level View/Modify/Full permissions. Consumers do not receive View or Modify permission on the individual profiles associated with the asset. You will have to assign permissions to the asset's individual profiles manually.
- If you use this action in a policy that applies to one (and only one) type of asset, you can set the asset's top-level View/Modify/Full permissions and also the View/Modify permissions on its individual profiles.

The permission settings you specify in this action will either replace or be merged with the asset's existing settings, depending on how you set the `Remove Existing Permission` parameter.

If you set `Remove Existing Permission` to true, the permission settings specified in the action *completely replace* the asset's current settings. That is, the asset's previous instance-level settings are completely cleared and the permissions specified by the action are set.

For example if an asset's initial permission settings are as follows:

```
USER A    Full
USER B    Full
```

And you specify the following permissions (with `Remove Existing Permission` set to true):

```
USER A    Full
GROUP X   Modify
```

The resulting permissions on the asset will be:

```
USER A    Full
GROUP X   Modify
```

If you set `Remove Existing Permission` to false, the permission settings specified by this action are added to the asset's current settings. So, for example, if an asset has the following permission settings:

```
USER A    Full
USER B    View
```

And you specify the following permissions (with `Remove Existing Permission` set to false):

```
USER A    Modify
USER B    Full
GROUP X   Modify
```

The resulting permissions on the asset will be:

```
USER A    Full
USER B     Full
GROUP X    Modify
```

Note: The instance-level permissions that this action assigns to a user does not affect any role-based permissions that the user might already have. For example, if user ABC has `Manage Assets` permission for an organization, and that user also happens to be a member of a group to which this action assigns instance-level permissions, user ABC's `Manage Assets` permission will override the permission settings that this action assigns to him or her.

Event Scope

OnConsumerRegistration

Object Scope

Asset (any type)

Input Parameters

`Consumer Asset Profile Permission`	*Object* The instance-level permissions that are to be assigned to the users and/or groups (specified in the consumer registration request) for the requested asset.
`Remove existing permission`	*Boolean* Specifies whether the permission settings in the `Consumer Asset Profile Permission` parameter replace the existing permission settings or whether they are combined with the existing settings.

Set Instance and Profile Permissions

Sets instance-level permissions on an asset. You can use this action to set top-level View/Modify/Full permissions on an entire asset and to set View/Modify permissions on individual profiles within an asset.

Note: You use this action to set permissions on assets only. To set permissions on policies, you must use the "Set Permissions" on page 81 action. If you want to assign asset permissions to consumers during the consumer registration process, use the "Set Consumer Permission" on page 77 action.

Be aware that the behavior of this action varies depending on the policy's object scope.

- If you use this action in a policy that applies to multiple asset types, you can only use it to set the asset's top-level View/Modify/Full permissions. Users do not receive View or Modify permission on the individual profiles associated with the asset. You have to assign permissions to the asset's individual profiles manually.

- If you use this action in a policy that applies to one (and only one) type of asset, you can use it to set the asset's top-level View/Modify/Full permissions and also the View/Modify permissions on its individual profiles.

The permission settings you specify in this action will either replace or be merged with the asset's existing settings, depending on how you set the `Remove Existing Permission` parameter.

If you set `Remove Existing Permission` to true, the permission settings specified in the action *completely replace* the asset's current settings. That is, the asset's previous instance-level settings are completely cleared and the permissions specified by the action are set.

For example if an asset's initial permission settings are as follows:

```
USER A    Full
USER B    Full
```

And you specify the following permissions (with `Remove Existing Permission` set to true):

```
USER A    Full
GROUP X   Modify
```

The resulting permissions on the asset will be:

```
USER A    Full
GROUP X   Modify
```

If you set `Remove Existing Permission` to false, the permission settings specified by this action are added to the asset's current settings. So, for example, if an asset has the following permission settings:

```
USER A    Full
USER B    View
```

And you specify the following permissions (with `Remove Existing Permission` set to false):

```
USER A    Modify
USER B    Full
GROUP X   Modify
```

The resulting permissions on the asset will be:

```
USER A    Full
USER B    Full
GROUP X   Modify
```

Note: The instance-level permissions that this action assigns to a user does not affect any role-based permissions that the user might already have. For example, if user ABC has `Manage Assets` permission for an organization, and that user also happens to be a member of a group to which this action assigns instance-level permissions, user ABC's `Manage Assets` permission will override the permission settings that this action assigns to him or her.

Event Scope

PostCreate

PreStateChange
PostStateChange
OnTrigger

Object Scope

Asset (any type)

Input Parameters

`User/ Group Asset Permission`	*Object Array* An array of permission settings. Each setting in the array identifies one individual user or one group and specifies the permissions for that user or group. If you specify multiple groups in this array and a user is a member of more than one group, the user will receive the permissions of all those groups combined. For example, if you assign Modify permission to Group A and Full permissions to Group B, users that are members of both groups will get Full permission on the object.
`Remove existing permission`	*Boolean* Specifies whether the permission settings in the parameters `User/Group Asset Permission`, `Propagate permissions to dependent objects` and `Propagate profile permissions` replace the existing permission settings or whether they are combined with the existing settings.
`Propagate permissions to dependent objects`	*Boolean* Specifies whether the access permissions defined for the asset instance will be automatically propagated to all dependent objects. For example, a Service asset can refer to a WSDL which in turn can refer to one or more XML Schema assets, and when you set this parameter to `yes`, changes in the access permissions in the Service asset will be propagated to all of these dependent assets.
`Propagate profile permissions`	*Boolean* Specifies whether the profile permissions defined for the asset instance will be automatically propagated to all dependent assets of the same type. The restriction concerning the asset type arises because different asset types can have different sets of profiles. The use of this parameter is restricted to the following asset types: ■ Service ■ XML Schema

Set Permissions

Grants View, Modify or Full permissions to specified users (or to groups of users) for a policy.

Note: You use this action to set permissions on policy objects. To set permissions on catalog assets, you must use "Set Instance and Profile Permissions" on page 78.

Be aware that the permission settings you specify in the action will either replace or be merged with the object's existing settings, depending on how you set the `Remove Existing Permission` parameter.

If you set `Remove Existing Permission` to true, the permission settings specified in the action will completely replace the object's current settings. That is, the action will clear the object's existing permission settings and replace them with the permissions you specify.

For example if a policy's initial permission settings were as follows:

```
USER A     Full
USER B     Full
GROUP ABC  Full
```

And you were to specify the following permissions with `Remove Existing Permission` set to true:

```
USER A     Full
GROUP X    Modify
```

The resulting permissions on the asset would be:

```
USER A     Full
GROUP X    Modify
```

If you set `Remove Existing Permission` to false, the permission settings specified in the action are *added to* the object's current settings. That is, the action will merge the new permission settings with the object's existing settings. For example, if an asset had the following permission settings:

```
USER A     Full
USER B     View
GROUP ABC  View
```

And you were to specify the following permissions with `Remove Existing Permission` set to false:

```
USER A     Modify
USER B     Full
GROUP X    Modify
```

The resulting permissions on the asset will be:

```
USER A     Full
USER B     Full
GROUP X    Modify
GROUP ABC  View
```

Note: The instance-level permissions that this action assigns to a user will not affect any role-based permissions that the user might already have. For example, if user ABC has `Manage Policies` permission for an organization and that user also happens to be a member of a group to which this action assigns instance-level permissions, user ABC's `Manage Policies` permission will override the permission settings that this action assigns to him or her.

Event Scope

PostCreate
PreStateChange
PostStateChange
OnTrigger

Object Scope

This action can be enforced on the following object types.

Policy

Input Parameters

`User/Group Permission`	*Object Array* An array of permission settings. Each setting in the array identifies one individual user or one group and specifies the permissions for that user or group. If you specify multiple groups in this array and a user is a member of more than one group, the user will receive the permissions of all those groups combined. For example, if you assign Modify permission to Group A and Full permissions to Group B, users that are members of both groups will get Full permissions on the object.
`Remove existing permission`	*Boolean* Specifies whether the permission settings in the `Users and Groups` parameter replace the existing permission settings or whether they are combined with the existing settings.
`Propagate permissions to dependent objects`	*Boolean* Specifies whether the access permissions defined for the asset instance will be automatically propagated to all dependent objects. For example, a Service asset can refer to a WSDL which in turn can refer to one or more XML Schema assets, and when you set this parameter to `yes`, changes in the access permissions in the Service asset will be propagated to all of these dependent assets.

Set Profile Permissions

This action sets an asset's profile permissions for the users/groups specified without setting the asset's instance level permissions.

The users/groups specified in the parameter should have view or modify instance level permission on the asset.

Event Scope

PostCreate
PreStateChange
PostStateChange
OnTrigger

Object Scope

Asset (any type)

Input Parameters

`User/Group Permission`	*Object Array* An array of permission settings. Each setting in the array identifies one individual user or one group, the specified profile and the view/modify permissions for that user or group for the profile.
`Remove existing permission`	*Boolean* Specifies whether the permission settings in the `User/Group Permission` parameter replace the existing permission settings or whether they are combined with the existing settings.

Set State

Initiates a lifecycle state change for a lifecycle model, policy, asset or Process object.

When you use this action, be aware that:

- The state change performed by this action will trigger PreStateChange or PostStateChange policies if such policies exist for the specified state change.
- When CentraSite executes this action at enforcement time, it attempts to change the target object to the state you have specified. If this state is not a valid transition from the object's current state, the action will fail.
- If the target object is already in the specified state at enforcement time, this action does nothing. It does not initiate a state change. It simply exits and returns a successful completion code (i.e., this condition is not considered an error).

Event Scope

PostStateChange
OnTrigger
OnConsumerRegistration

Object Scope

Lifecycle Model
Policy
Asset (any type)
Process object

Input Parameters

`Change State To`	*String* The value to which you want to set the object's state.

Set View Permission for Service and Service Related Object to Everyone Group

Grants the View permission on a given service to the Everyone group. When permission is given to Everyone, all users, including guests, are able to view the service and its related interface, operation and binding objects. This policy action enables UDDIv2 clients to access the service without providing an authtoken.

This action is included in the *UDDIv2 Inquiry Policy* policy that is installed with CentraSite. This policy executes when a service or virtual service is created. This policy is disabled by default.

Event Scope

PostCreate
OnTrigger

Object Scope

Assets

Input Parameters

None.

Send Email Notification to Watchers

Sends an email notification to the watchers for an asset who are specific users asked to be notified for any modifications on that particular asset.

Note: This action is applicable to the CentraSite Business UI.

Event Scope

PostUpdate
PostDelete
OnTrigger

Object Scope

Asset (any type)

Input Parameters

`Email Template`	*Email Template* Specifies the template that is to be used to generate the body of the email message. **Note:** You can use the predefined template, NotifyUsersOnUpdate.html, as your email template if you do not want to create an email template of your own.
`Format`	*String* Specifies whether the mail message is formatted as HTML or plain text.

UnClassify

Removes specified taxonomy categories from an object.

You can use this action to unclassify an object generally or specifically. If you want to unclassify an object by removing from it all categories for an entire taxonomy, use the Taxonomies parameter to specify the taxonomy name. If you want to unclassify an object by removing just one particular category from its classification attributes, you use the Categories parameter to specify a specific category name. Both parameters can be used in the same action.

This action is executed against all classification attributes in the target object.

If the target object is not classified by any of the taxonomies or classifiers specified in the Taxonomies or Categories parameters, the action simply exits and returns a successful completion code. This condition is not considered to be an error.

Event Scope

PostStateChange
OnTrigger
OnConsumerRegistration

Object Scope

This action can be enforced on any object type that the policy engine supports.

Input Parameters

`Taxonomies`	*String Array* The names of the taxonomies whose categories are to be removed from the target object.
`Categories`	*String Array* The names of specific categories that are to be removed from the target object.

UnPublish from API-Portal

Removes specified API metadata from the API-Portal repository.

Event Scope

- PreDelete
- PostDelete
- PreStateChange
- PostStateChange
- OnTrigger

Object Scope

- Service
- XML Service
- REST Service
- Virtual Service
- Virtual XML Service
- Virtual REST Service

Input Parameters

`API-Portal`	*Optional. String. Array.* The name of the API-Portal from that API-repository the API metadata would be removed. **Note:** However, if this action is to be executed in a different event other than OnTrigger, for example,

prestatechange or poststagechange, that is not provided by default, you *must* specify a value for this field.

Validate Attribute Value

Validates the value of a specified attribute in an organization, user or asset against a list of allowed values.

Event Scope

PreStateChange
PreDelete
OnTrigger
OnConsumerRegistration

Object Scope

Organization
User
Asset (any type)

Input Parameters

`Attribute Name`	The name of the attribute that you want this action to test. The attribute's data type can be Boolean, Date and Time, Duration, Email, IP Address, Multiline String, Number, and URL/URI. **Note:** `Attribute Name` must be a non-arrayed attribute.
`Possible Attribute Value`	*String Array* An array of regular expression String values. If the value of the attribute specified in `Attribute Name` matches any entry in `Possible Attribute Values`, the action succeeds. The regular expressions you specify in `Possible Attribute Values` must support the regular expression specification for Java. The data types of possible attribute values can be Boolean, Date and Time, Duration, Email, IP Address, Multiline String, Number, and URL/URI. You can include substitution tokens in this parameter to incorporate data from the target object on which the policy is acting. For a list of the allowed tokens, see the list of Substitution Tokens shown in the Send Email Notification action.

Validate Classification

Checks whether an object is classified by a given taxonomy or taxonomy category. This action examines all classification attributes in the target object.

If you just want to check that the target object has been classified by a given taxonomy, simply specify the taxonomy in the Taxonomies parameter. Leave the Categories parameter empty. The action will succeed if the object is classified by *any* category in the taxonomy (i.e., the action succeeds if the object includes at least one Classification attribute whose value represents a category that belongs the specified taxonomy).

If you want to check that the target object has been classified by a specific category in a taxonomy, specify the exact category in the Categories parameter. Leave the Taxonomies parameter empty. The action will succeed only if the object has been classified by the exact category you specify (i.e., the object includes at least one Classification attribute whose value is set to that specific category).

If you specify multiple taxonomies and categories in the Taxonomies and Categories parameters, be aware that action will succeeds if the target object is classified according to *any* taxonomy specified in the Taxonomies parameter or *any* category specified in the Categories parameter. If you need to verify that an object has been classified by several different taxonomies or categories, you must test for each required taxonomy or category using a separate Validate Classification action.

Event Scope

PreStateChange
PreDelete
OnTrigger
OnConsumerRegistration

Object Scope

This action can be enforced on any object type that the policy engine supports.

Input Parameters

`Taxonomies`	*String Array* The names of the taxonomies by which the object must be classified.
`Categories`	*String Array* The names of specific taxonomy nodes by which the target object must be classified.

Validate Description

Validates the description of an object against a given pattern string.

Event Scope

PreCreate
PreStateChange
OnTrigger

Object Scope

This action can be enforced on any object type that the policy engine supports.

Input Parameters

`Allowed Description Pattern`	*String* Specifies a regular expression that the description must satisfy. The regular expressions you specify in `Allowed Description Pattern` must support the regular expression specification for Java. The regular expression can include substitution tokens to incorporate data from the target object on which the policy is acting. For a list of the allowed tokens, see the list of Substitution Tokens shown in the Send Email Notification action.

Validate Lifecycle Model Activation

Verifies that a lifecycle model is ready to be activated by checking that the following conditions exist for the lifecycle model:

- That the lifecycle model is associated with at least one object type.
- That the object types associated with the lifecycle model are not already assigned to an active lifecycle model in your organization. (This check ensures that, within your organization, each object type is associated with no more than one lifecycle model.)

The action will not succeed unless both conditions are satisfied.

You should include this action in any policy that is triggered by a lifecycle state change that subsequently activates the lifecycle model. Executing this action before the state change occurs ensures that the state change (and subsequent activation) will not occur unless the lifecycle model is capable of being activated.

This action is executed by the default *Validate Lifecycle Activation* policy that is installed with CentraSite. The Validate Lifecycle Activation policy executes on the PreStateChange event that occurs when a lifecycle model switches to the Productive lifecycle state. The Validate Lifecycle Activation action in this policy ensures that a lifecycle model is not switched to the Productive state (and consequently, activated) unless the model has been properly associated with one or more object types.

Event Scope

PreStateChange

Object Scope

Lifecycle Model

Input Parameters

None.

Validate Name

Validates the name of an object against a given pattern string.

Event Scope

PreCreate
PreStateChange
OnTrigger

Object Scope

This action can be enforced on any object type that the policy engine supports.

Input Parameters

`Allowed Name Pattern`	*String* Specifies a regular expression that the object name must satisfy. The regular expressions you specify in `Allowed Name Pattern` must support the regular expression specification for Java. The regular expression can include substitution tokens to incorporate data from the target object on which the policy is acting. For a list of the allowed tokens, see the list of Substitution Tokens shown in the Send Email Notification action.

Validate Namespace

Checks that the targetnamespace attribute in a Web Service or XML Schema matches one of the valid namespaces in a given list.

Event Scope

PreCreate
PreStateChange

OnTrigger

Object Scope

XML Schema
CEP Event Type
Service
Virtual Service
REST Service
Virtual REST Service
XML Service
Virtual XML Service

Input Parameters

`Allowed Namespaces`	*String Array* An array of regular expressions representing the valid namespaces. For this action to succeed, the value of the targetnamespace attribute in the service WSDL or XML schema must satisfy one of the regular expressions in the array. The regular expressions you specify in `Allowed Namespaces` must support the regular expression specification for Java. The regular expression can include substitution tokens to incorporate data from the target object on which the policy is acting. For a list of the allowed tokens, see the list of Substitution Tokens shown in the Send Email Notification action.

Validate Policy Activation

Verifies that a policy is ready to be activated by checking that the following conditions exist for the policy:

- That all of the required parameters in the policy's action list have been set.
- That all of the actions in the action list are supported by the policy's specified scope. That is, the policy does not contain any action whose scope includes an object type or event type that is outside the scope of the policy itself.
- That a policy that contains one or more WS-I actions contains *only* WS-I actions.
- That a policy that executes on a PreStateChange or PostStateChange specifies the lifecycle states that will trigger the policy.
- Whether a previous version of the policy is already active, and if so, it verifies that the policy can be switched to a state in which it is retired or superseded.

The action will not succeed unless all conditions are satisfied.

You should include this action in any policy that is triggered by a lifecycle state change that subsequently activates the policy. Executing this action before the state change occurs ensures that state change (and subsequent activation) will not occur unless the policy is capable of being activated.

This action is executed by the default *Validate Policy Activation* policy that is installed with CentraSite. The Validate Policy Activation policy executes on the PreStateChange event that occurs when a policy switches to the Productive lifecycle state. The Validate Policy Activation action in this policy ensures that a policy is not switched to the Productive state (and consequently activated) unless the policy's action parameters have been set.

Event Scope

PreStateChange

Object Scope

Policy

Input Parameters

None.

Validate Policy Deactivation

Verifies that a policy is not currently “in-progress” (that is, undergoing execution) and can therefore be successfully deactivated. If the policy is in-progress when this action is executed, this action will fail.

You should include this action in any policy that is triggered by a lifecycle state change that subsequently deactivates the policy. Executing this action before the state change occurs helps ensure that the stage change (and subsequent policy deactivation) will not take place if the target policy is in-progress.

Note: A policy that initiates an approval workflow is considered to be “in-progress” until the required approvals are obtained for the workflow. Therefore, if the Validate Policy Deactivation action is triggered for a policy that is associated with one or more pending approval workflows, the action will fail.

This action is executed by the default *Validate Policy Deactivation* policy that is installed with CentraSite. The Validate Policy Deactivation policy executes on the PreStateChange event that occurs when a policy switches to the Revising or Retired state. The Validate Policy Deactivation action in this policy ensures that a policy is not switched to the Revising or Retired state (and consequently, deactivated) while it is undergoing execution.

Event Scope

PreStateChange

Object Scope

Policy

Input Parameters

None.

Validate Service Binding

Checks that a Web Service supports the specified bindings.

Event Scope

PreCreate
PreStateChange
OnTrigger

Object Scope

Service
Virtual Service

Input Parameters

`Binding Types`	*String Array* An array containing the list of binding types that the Web Service must support. The action will succeed only if the Web service supports all of the bindings specified in `Binding Types`.

Validate State

Validates the current state of a lifecycle model, policy or asset against a given list of states.

Event Scope

PreDelete
OnTrigger
OnConsumerRegistration

Object Scope

Lifecycle Model
Policy
Asset (any type)

Input Parameters

`Allowed States`	*String Array* An array that specifies the states for which you want the target object checked. If the state of the object matches any entry specified in `Allowed States`, the action succeeds.

Validate WSDL Size

Checks the size of the WSDL document associated with a Web Service to ensure it falls within a specified range.

Event Scope

PreCreate
PreStateChange
OnTrigger

Object Scope

Service
Virtual Service

Input Parameters

`WSDL Size`	*Number* The size limit (expressed in the units specified by the `Size Unit` parameter, below.)
`Comparator`	*String* A relational operator that specifies how the size of the WSDL document is to be compared to the value in `WSDL Size`.
`Size Unit`	*String* The units in which `WSDL Size` is expressed. Valid values are 'KB' (for Kilobytes) or 'MB' (for Megabytes).

webMethods REST Publish

Creates a REST service from the published IS service interface object.

The action is included in the *webMethods REST Publish* policy that is installed with CentraSite. This policy automatically executes when the webMethods Designer publishes an IS Service Interface object.

Important: This IS Service Interface object should be classified under the concept called WMAssetType -> Integration Server Asset -> TypeOfIntegrationServiceInterface -> REST Service.

Event Scope

Post-Create
Pre-Update

Object Scope

IS Service Interface

Input Parameters

None.

3 Access via UDDI

■ Overview of the UDDI Standard 98
■ Summary of UDDI Support in CentraSite 99
■ CentraSite UDDI Architecture 100
■ UDDI Representation of the Object Model 106
■ Configuring the UDDI Environment 115
■ Predefined Value Sets 126
■ Predefined tModels 127
■ UDDI V3 APIs 129
■ Using Third-Party IDE Tools with CentraSite 135
■ UDDI Extensions 137

Overview of the UDDI Standard

UDDI (Universal Description Discovery & Integration) is a platform-independent standard maintained by the OASIS consortium. The standard describes a Service Oriented Architecture (SOA) registry and its interfaces. UDDI allows clients to discover registered businesses (organizations or providers) and the web services they provide. UDDI also provides programming interfaces to create and update the stored registry information.

The current release of UDDI is V3.0.2. In this document, the UDDI version number is abbreviated to `V3`.

The UDDI data model defines the following entity types:

Entity Type	Description
businessEntity	Represents a business.
businessService	Represents one or more web services provided by a business.
bindingTemplate	Describes how to use a web service.
tModel	Categorizes a web service type.
publisherAssertion	Represents a relationship between two business entities.

UDDI defines a set of APIs for accessing and modifying the data stored in the registry.

These APIs include:

- **UDDI Inquiry**

  This API allows you to search for registry entries and retrieve information about them. The search mechanism allows the use of browse patterns (i.e. wildcards) to be used, so that a set of matching business entities can be returned. The use of a subsequent so-called drill-down pattern allows information to be retrieved from a business entity.

- **UDDI Publication**

  This API allows you to add entries to the registry or modify existing entries.

- **UDDI Security**

  This API determines which security settings apply for a registry entity.

- **Custody and Ownership Transfer**

This API set is for transferring UDDI objects between multiple nodes of a UDDI registry and transferring the ownership of UDDI objects between users.

- **Replication**

  This API set is for synchronizing the data of multi-node UDDI registries.

In addition, the UDDI specification defines the following client API set:

- Value Set

Summary of UDDI Support in CentraSite

CentraSite provides support for UDDI V3. The full list of features of UDDI V3 is available at the OASIS web site mentioned above.

The features currently not supported in CentraSite are as follows:

- **Value Set API Set**

  This API is not supported.

- **Replication**

  Replication is not supported.

- **Publishing Across Multiple Registries**

  This feature is not supported.

- **UDDI Policies**

  UDDI policies are not supported.

  Note: UDDI policies are not the same as web service policies.

- CentraSite supports ownership transfer but not inter-node custody transfer.
- **Multi-Version Support**

  The following features are not supported:

  - Migrating version 2 keys to a version 3 registry
  - Multiple `xml:lang` attributes of the same language
  - Supporting external value set providers across versions
  - White space handling
  - Multiple `overviewDoc` data
  - Multiple `personName` data

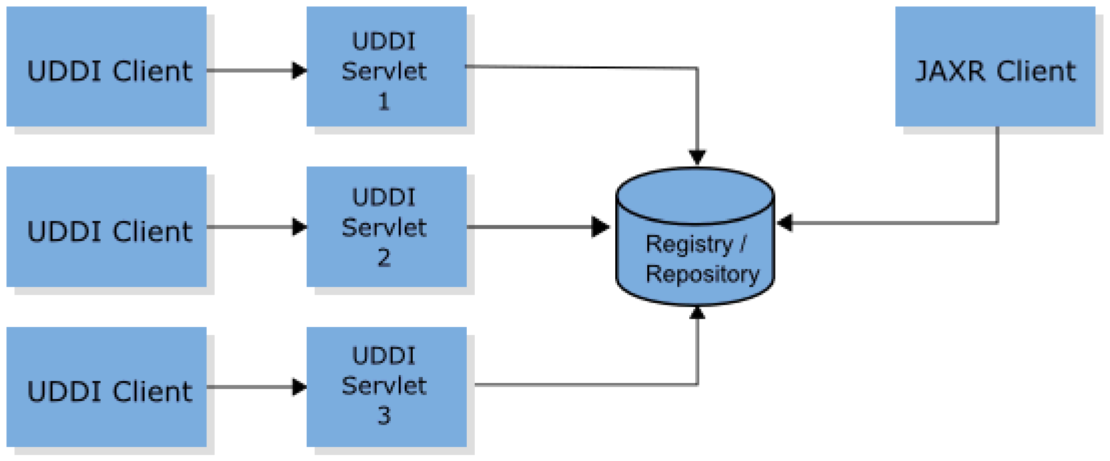
CentraSite UDDI Architecture

Overview

CentraSite behaves like a UDDI registry, as described in the UDDI specification. The main components of CentraSite's UDDI environment are:

- The CentraSite Registry/Repository, in which the UDDI objects are stored.
- One or more UDDI servlets running on different application servers. Each UDDI servlet implements the web services of the UDDI API sets.

The following figure illustrates the multiple UDDI servlet scenario, in which multiple UDDI clients and a JAXR client interact with a single CentraSite registry/repository:



Although there can be multiple UDDI servlets, a CentraSite installation is a single-node UDDI registry. Each UDDI servlet just provides an alternative endpoint for the UDDI web services.

When UDDI data is stored in the CentraSite Registry/Repository, it is mapped to a data model that is common for JAXR and UDDI clients. The data model is an XML representation of JAXR data. Since this representation is also used to store data from the JAXR API, UDDI and JAXR clients act on the same data. Note that JAXR instance-based security is the basis of the CentraSite UDDI security, so changes in the JAXR instance-based security may affect UDDI security.

Client Access via UDDI

UDDI clients can access the CentraSite registry using the following URLs.

The URL for the inquiry API is: `http://<hostname>:53307/UddiRegistry/inquiry`, where *<hostname>* is the name of the host machine. For example, if the

UDDI client is running on the same machine as the UDDI servlet, the URL is `http://localhost:53307/UddiRegistry/inquiry`.

The URL for the publish API is: `http://<hostname>:53307/UddiRegistry/publish`.

Every UDDI servlet in a multiple UDDI servlet environment has these endpoints.

Localization

Localization for UDDI means that the error messages are localized. These messages are given in the content of an `errorInfo` element in the disposition report. Following is an example disposition report:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <soapenv:Fault>
      <faultcode>Client</faultcode>
      <faultstring>Client Error</faultstring>
      <faultactor />
      <detail>
      <dispositionReport generic="3.0" xmlns="urn:uddi-org:api_v3">
        <result errno="10120">
          <errInfo errCode="E_authTokenRequired">
            The authentication token value dummy passed in the authInfo
            argument of the UDDI request is not valid.
          </errInfo>
        </result>
      </dispositionReport>
      </detail>
    </soapenv:Fault>
  </soapenv:Body>
</soapenv:Envelope>
```

Modeling of the Node Business Entity

The modeling of the Node Business Entity is based on the Recommended Modeling of Node Business Entity in the UDDI 3.0.2 standard.

Key Generator tModel

The UDDI objects needed for the self-modeling of the node business entity have keys defined internally by CentraSite that belong to the key partition that is owned by the key generator:

```
<tModel
  tModelkey="...:keyGenerator"
  xmlns="urn:uddi-org:api_v3">
    <name xml:lang="en-US">centrasite-node-com:keyGenerator</name>
    <description xml:lang="en-US">
        Key generator for self registering node business entity
    </description>
    <categoryBag>
       <keyedReference
          tModelkey="...:keyGenerator"
          keyName="uddi-org:types:keyGenerator"
          keyValue="keyGenerator"/>
     </categoryBag>
  </categoryBag>
</tModel>
```

The key generator is preloaded in the UDDI registry.

Node Business Entity

The node business entity that is preloaded into the Registry/Repository looks like this:

```
<businessEntity
  businessKey="...">
    <name xml:lang="en-US">NodeBusinessEntity</uddi:name>
    <description xml:lang="en-US">
        Node Business Entity of the CentraSite UDDI registry
    </description>
    <categoryBag>
       <uddi:keyedReference
          keyName=""
          keyValue="node"
          tModelkey="...:keyGenerator"/>
    </categoryBag>
</businessEntity>
```

The node business entity references certain `businessService` objects which reflect the UDDI API sets. The first UDDI registry interacting with the Registry/Repository adds the `businessService` objects. It also adds the `bindingTemplate` objects pointing to the API set's endpoints offered by the UDDI registry. Every additional UDDI registry in a multiple UDDI servlet environment adds `bindingTemplate` objects pointing to the additional endpoints where the UDDI services can be called. The tModels referenced by the `businessService` objects are preloaded into the UDDI registry. Due to the fact that multiple `bindingTemplate` objects are defined for each UDDI registry, the `bindingTemplate` objects get node generated keys.

The web services that implement the UDDI API sets are themselves registered in the UDDI registry. For CentraSite, only the supported API sets are reflected by this self-registration. These are:

- Inquiry API set
- Publication API set
- Security Policy API set
- Custody and Ownership Transfer API Set

Inquiry Service

```
<businessService
  serviceKey="uddi:centrasite.node.com:service_inquiry"
  businessKey="..."
  xmlns="urn:uddi-org:api_v3">
    <name xml:lang="en-US">UDDI Inquiry Services</name>
    <description xml:lang="en-US">Web Service supporting UDDI Inquiry APIs
    </description>
    <bindingTemplates>
       <bindingTemplate
          bindingKey="..."
          serviceKey="uddi:centrasite.service.com:inquiry">
            <description xml:lang="en-US">
               This binding supports the UDDI Programmer's API Specification
               For inquiry
            </description>
            <accessPoint useType="endPoint">
               http://localhost:53307/UddiRegistry/inquiry
            </accessPoint>
            <tModelInstanceDetails>
              <tModelInstanceInfo
                tModelkey="...:keyGenerator">
                  <description xml:lang="en-US">
                      This access point supports the UDDI Version 2.0
                      Programmer's API Specification for inquiry
                  </description>
              </tModelInstanceInfo>
           </tModelInstanceDetails>
       </bindingTemplate>
       <bindingTemplate
         bindingKey="..."
         serviceKey="uddi:centrasite.node.com:service_inquiry">
            <description xml:lang="en-US">
               This binding supports the UDDI Programmer's API Specification
               for inquiry
            </description>
            <accessPoint useType="endPoint">
               http://localhost:53307/UddiRegistry/inquiry
            </accessPoint>
            <tModelInstanceDetails>
               <tModelInstanceInfo tModelkey="...:keyGenerator">
                  <description xml:lang="en-US">
                      This access point supports the UDDI Version 3.0
                      Programmer's API Specification for inquiry
                  </description>
               </tModelInstanceInfo>
            </tModelInstanceDetails>
       </bindingTemplate>
    </bindingTemplates>
</businessService>
```

Publish Service

```
<businessService
  serviceKey="uddi:centrasite.node.com:service_publish"
  businessKey="..."
  xmlns="urn:uddi-org:api_v3">
    <name xml:lang="en-US">UDDI Publish API Services</name>
    <description xml:lang="en-US">
        Web Service supporting UDDI specifications
    </description>
    <bindingTemplates>
        <bindingTemplate
          bindingKey="..."
          serviceKey="uddi:centrasite.node.com:service_publish">
            <description xml:lang="en">
                This binding supports the UDDI Programmer's API Specification
                for publication
            </description>
            <accessPoint useType="endPoint">
                http://localhost:53307/UddiRegistry/publish
            </accessPoint>
            <tModelInstanceDetails>
                <tModelInstanceInfo
```

```
                   tModelkey="...:keyGenerator">
                     <description xml:lang="en">
                         This binding supports the UDDI Version 2.0 Programmer's
                         API Specification for publication
                     </description>
                 </tModelInstanceInfo>
             </tModelInstanceDetails>
         </bindingTemplate>
         <bindingTemplate
           bindingKey="uddi:centrasite.node.com:binding_publish_v3"
           serviceKey="uddi:centrasite.node.com:service_publish">
             <description xml:lang="en">
                 This binding supports the UDDI Programmer's API Specification
                 for publication
             </description>
             <accessPoint useType="endPoint">
                 http://localhost:53307/UddiRegistry/publish
             </accessPoint>
             <tModelInstanceDetails>
                 <tModelInstanceInfo tModelkey="...:keyGenerator">
                     <description xml:lang="en">
                         This binding supports the UDDI Version 3.0 Programmer's
                         API Specification for publication
                     </description>
                 </tModelInstanceInfo>
             </tModelInstanceDetails>
         </bindingTemplate>
     </bindingTemplates>
</businessService>
```

Security Service

```
<businessService
  serviceKey="uddi:centrasite.node.com:service_security"
  businessKey="..."
  xmlns="urn:uddi-org:api_v3">
    <name xml:lang="en">UDDI Security Service</name>
    <description xml:lang="en-US">
       Web Service supporting UDDI Security API
    </description>
    <bindingTemplates>
        <bindingTemplate
          bindingKey="..."
          serviceKey=" uddi:centrasite.node.com:service_security">
            <description xml:lang="en">
                This binding to authenticate with the UDDI services using the
                UDDI Security API.
            </description>
            <accessPoint useType="endPoint">
               http://localhost:53307/UddiRegistry/publish
            </accessPoint>
            <tModelInstanceDetails>
                <tModelInstanceInfo tModelkey="...:keyGenerator">
                    <description xml:lang="en">
                         This binding's supports the UDDI v3 Security API.
                    </description>
                </tModelInstanceInfo>
            </tModelInstanceDetails>
        </bindingTemplate>
    </bindingTemplates>
</businessService>
```

Custody and Ownership Transfer Service

```
<businessService
  serviceKey="uddi:centrasite.node.com:service_ownership_transfer"
  businessKey="..."
  xmlns="urn:uddi-org:api_v3">
    <name xml:lang="en">UDDI Custody and Ownership Transfer API</name>
    <description xml:lang="en-US">
       Web Service providing partly support for the UDDI Custody and Ownership
       Transfer API
    </description>
    <bindingTemplates>
        <bindingTemplate
          bindingKey="uddi..."
          serviceKey=" uddi:centrasite.node.com:service_ownership_transfer">
            <description xml:lang="en-US">
                This binding provides partly support for the UDDI Custody and
                Ownership Transfer API.
            </description>
            <accessPoint useType="endPoint">
               http://localhost:53307/UddiRegistry/publish
            </accessPoint>
            <tModelInstanceDetails>
                <tModelInstanceInfo
                  tModelkey="...:keyGenerator">
                    <description xml:lang="en-US">
                        This binding provides partly support for the UDDI
                        Custody and Ownership Transfer API
                    </description>
                </tModelInstanceInfo>
            </tModelInstanceDetails>
        </bindingTemplate>
    </bindingTemplates>
</businessService>
```

WSDL

The referenced tModels refer to the WSDL file shown below. Each port, specified by the `<port>` element, specifies an access point. The WSDL representation is a description of the web services that are provided by the UDDI servlets.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="UDDI_API_V3"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:api_v3_binding="urn:uddi-org:api_v3_binding"
  xmlns:tns="urn:uddi-org:api_v3"
  targetNamespace="urn:uddi-org:api_v3">

    <documentation>
    UDDI V3 Security, Publication, Inquiry and Transfer APIs.
    </documentation>

    <import
      namespace="urn:uddi-org:api_v3_binding"
      location="http://uddi.org/wsdl/uddi_api_v3_binding.wsdl"/>
    <import
      namespace="urn:uddi-org:api_v3_binding"
      location="http://uddi.org/wsdl/uddi_custody_v3_binding.wsdl"/>
    <service name="UDDI_Security_SoapService">
```

```
        <port
          name="UDDI_Security_PortType"
          binding="api_v3_binding:UDDI_Security_SoapBinding">
            <soap:address
              location="http://localhost:53307/UddiRegistry/publish"/>
        </port>
    </service>
    <service name="UDDI_Publication_SoapService">
        <port
          name="UDDI_Publication_PortType"
          binding="api_v3_binding:UDDI_Publication_SoapBinding">
            <soap:address
              location="http://localhost:53307/UddiRegistry/publish"/>
        </port>
    </service>
    <service name="UDDI_Inquiry_SoapService">
        <port
          name="UDDI_Inquiry_PortType"
          binding="api_v3_binding:UDDI_Inquiry_SoapBinding">
            <soap:address
              location="http://localhost:53307/UddiRegistry/inquiry"/>
        </port>
    </service>
    <service name="UDDI_Ownership_Transfer_SoapService">
        <port
          name="UDDI_Inquiry_PortType"
          binding="api_v3_binding:UDDI_Inquiry_SoapBinding">
            <soap:address
              location="http://localhost:53307/UddiRegistry/inquiry"/>
        </port>
    </service>
</definitions>
```

The WSDL defines the proper endpoints of each web service. Each UDDI registry adds a port pointing to its endpoints. The first UDDI registry also inserts the service elements. The WSDL file can be accessed via the URIs:

- `http://localhost:53307/UddiRegistry/inquiry?WSDL`
- `http://localhost:53307/UddiRegistry/publish?WSDL`

The WSDL is stored in the CentraSite repository.

UDDI Representation of the Object Model

Attributes

The purpose of attributes in CentraSite is to associate values, classifications and associations with names. For the UDDI representation in CentraSite, the attributes are grouped together in a `uddi:keyedReferenceGroup` holding all attributes of the UDDI object.

Key/Value Pair Attributes

Key/value pair attributes are represented in a `uddi:categoryBag`, as follows:

```
<categoryBag>
```

```
 <keyedReference
     tModelKey="uddi:uddi.org:categorization:general_keywords"
     keyName="Encryption Required" keyValue="false"/>
 <keyedReference
     tModelKey="uddi:uddi.org:categorization:general_keywords"
     keyName="Demo Available" keyValue="false"/>
</categoryBag>
```

Rich Text Attributes

For tModels and instanceDetails, Rich Text attributes are represented in a `uddi:overviewDoc` element, as follows.

```
…
<overviewDoc>
<description>
 Detailed Description
</description>
<overviewUrl useType="uddi:centrasite.com:attributes:richText">
 http://10.22.21.94:2020/...
</overviewUrl>
</overviewDoc>
…
```

For all other UDDI objects, especially `uddi:business` and `uddi:businessService`, a modified, UDDI-conformant `uddi:keyedReferenceGroup` approach provides more flexibility. However, the document referencing semantics cannot be interpreted by a standard UDDI client. Therefore, for Rich Text attributes a dedicated `uddi:keyedReferenceGroup` is introduced, as follows:

```
<categoryBag>
 <keyedReferenceGroup tModelKey="uddi:centrasite.com:attributes:richText">
   <keyedReference
         tModelKey="uddi:uddi.org:categorization:general_keywords"
         keyName="Detailed Description"
         keyValue="http://... "/>
 </keyedReferenceGroup>
</categoryBag>
```

The `uddi:tModelKey` is pointing to `uddi:uddi.org:categorization:general_keyword`s, to make the `uddi:keyName` meaningful. The `uddi:keyValue` points to the URL of the rich text. All `uddi:keyedReference`s representing a rich text attribute are stored in a single `uddi:keyedReferenceGroup`.

Document Attributes

CentraSite supports two different documents types:

- `reference`: This type is used for referencing documents that can be shared between registry objects. These are also called supporting documents.
- `contains`: This type is used for documents that belong exclusively to the definition of a registry object. Examples are WSDL and schema documents that define a Web service.

For tModels and instanceDetails, documents are represented in a `uddi:overviewDoc` element, as follows.

```
…
<overviewDoc>
<description>
 sample category/Documentation
</description>
<overviewUrl useType="uddi:centrasite.com:attributes:document:reference">
 http://10.22.21.94:2020/...
</overviewUrl>
</overviewDoc>
<overviewDoc>
<description>
 sample Asset File
</description>
<overviewUrl useType="uddi:centrasite.com:attributes:document:contains">
 http://10.22.21.94:2020/...
</overviewUrl>
</overviewDoc>
…
```

For all other UDDI objects, a modified `uddi:keyedReferenceGroup`-based approach is used, which conforms to the UDDI specification:

```
<categoryBag>
<keyedReferenceGroup
      tModelKey="uddi:centrasite.com:attributes:document:reference">
    <keyedReference
        tModelKey="uddi:uddi.org:categorization:general_keywords"
        keyName="sample category/Documentation"
        keyValue="http://10.22.21.94:2020/..."/>
</keyedReferenceGroup>
<keyedReferenceGorup
      tModelKey="uddi:centrasite.com:attributes:document:contains"
    <keyedReference
        tModelKey="uddi:uddi.org:categorization:general_keywords"
        keyName="Asset File" keyValue="http://10.22.21.94:2020/..."/>
 </keyedReferenceGroup>
</categoryBag>
```

For both document types (`reference` and `contains`), a separate `uddi:keyedReferenceGroup` is needed. The `uddi:tModelKey` of the `uddi:keyedReferenceGroup` specifies the document type. For each document type, a separate `uddi:tModel` is defined. Each `udddi:keydReference` represents a single document. The `uddi:tModelKey` points to `uddi:uddi.org:categorization:general_keyword`s, to add meaning to the `uddi:keyName`. The `uddi:keyName` specifies the location of the file, and the `uddi:keyValue` holds the URL of the document. All `uddi:keyedReference`s representing documents of one type are stored in a single `uddi:keyedReferenceGroup`.

Relationship Attributes

Relationships are represented in a `uddi:keyedReferenceGroup`, as follows:

```
<categoryBag>
<keyedReferenceGroup
            tModelKey="uddi:centrasite.com:attributes:relationShip">
   <keyedReference
             tModelKey="uddi:uddi.org:categorization:general_keywords"
             keyValue="uddi:3447…"
             keyName="Invokes"/>
   <keyedReference
             tModelKey="uddi:uddi.org:categorization:general_keywords"
             keyValue="uddi:3447…"
             keyName="RelatedTo"/>
 </keyedReferenceGroup>
</categoryBag>
```

The `uddi:tModelKey` indicates that a relationship is represented. Another `uddi:tModelKey` uses `uddi:uddi.org:categorization:general_keyword`s to add meaning to the `uddi:keyName`s. The `uddi:keyValue` points to the reference registry object, and the `uddi:keyName` specifies the type of the relationship. The available relationship types are determined by the attribute meta data. All `uddi:keyedReference`s representing relationships are stored in a single `uddi:keyedReferenceGroup`.

Metrics Definition

Following the approach proposed by the Governance Interoperability Framework, saving metrics information for service objects is accomplished by publishing a uddi:tModel. This means that a uddi:tModel holding the metrics information is published and attached to the given uddi:businessService.

Following is a simple uddi:tModel the metrics information:

```
 <tModel tModelKey="uddi:3d32ac10-5dd1-11da-88b8-51d47e6188b2" deleted="false"
xmlns="urn:uddiorg:api_v3">
<name>Metrics</name>
<description>Metrics of EchoAccessPoint</description>
<categoryBag>
<keyedReference
 tModelKey="…(key of object type taxonomy)"
 keyName="Metrics"
 keyValue="Metrics"/>

 <keyedReference
   tModelKey="uddi:centrasite.com:management:metrics:total.request.count"
   keyName="Count of hits"
   keyValue="14"/>

</categoryBag>
</tModel>
```

The uddi:keyedReference of the uddi:categoryBag hold the metrics information. The following metrics are supported.

- Total Request Count
- Success Request Count
- Fault Count
- Average Response Time
- Minimum Response Time
- Maximum Response Time
- Availability
- Service Liveliness

For each of the supported metrics an according taxonomy is defined that represents the according value set.

For attaching a metrics uddi:tModel to a given uddi:service, a uddi:keyedReference is used, which references the metrics-reference taxonomy. Following is an example of a uddi:businessService that references a metrics uddi:tModel:

```
<businessService serviceKey="uddi:1d233560-5dc8-11da-88b7-51d47e6188b2"
businessKey="uddi:a2b32100-5ac0-11da-8540-e2406020853d"
xmlns="urn:uddi-org:api_v3">
 <name>EchoHeadersService</name>
 <description>wsdl:type representing service</description>
 <bindingTemplates>
   <bindingTemplate bindingKey="uddi:1d25a660-5dc8-11da-88b7-51d47e6188b2"
     serviceKey="uddi:1d233560-5dc8-11da-88b7-51d47e6188b2">
     <description>wsdl:type representing port</description>
     <accessPoint useType="http://schemas.xmlsoap.org/soap/http">
       http://tracy:4400/sst/runtime.asvc/com.actional.soapstation.Echo
     </accessPoint>
     <tModelInstanceDetails>
       <tModelInstanceInfo
         tModelKey="uddi:1cd8bee0-5dc8-11da-88b7-51d47e6188b2">
         <instanceDetails>
           <instanceParms>EchoHeaders</instanceParms>
         </instanceDetails>
       </tModelInstanceInfo>
       <tModelInstanceInfo
         tModelKey="uddi:1cafda20-5dc8-11da-88b7-51d47e6188b2"/>
     </tModelInstanceDetails>
     <categoryBag>
       <keyedReference tModelKey="uddi:uddi.org:wsdl:types"
         keyName="uddi.org:wsdl:types" keyValue="port"/>
     </categoryBag>
   </bindingTemplate>
 </bindingTemplates>
 <categoryBag>
 <keyedReference
   tModelKey="uddi:centrasite.com:management:metrics:reference"
   keyName="Metrics"
     keyValue="uddi:3d32ac10-5dd1-11da-88b8-51d47e6188b2"/>
 <keyedReference tModelKey="uddi:uddi.org:xml:namespace"
   keyName="uddi.org:xml:namespace"
     keyValue="http://sanity/test"/>

 <keyedReference tModelKey="uddi:uddi.org:wsdl:types"
   keyName="uddi.org:wsdl:types" keyValue="service"/>

 <keyedReference tModelKey="uddi:uddi.org:xml:localName"
   keyName="uddi.org:xml:localName"
     keyValue="EchoHeadersService"/>
 </categoryBag>
</businessService>
```

Metrics Reference Taxonomy

The Metrics Reference taxonomy is for attaching metrics uddi:tModels to uddi:businessService. The uddi:tModel for defining the taxonomy looks as follows:

```
<tModel tModelKey="uddi: centrasite.com:management:metrics:reference">
 <name> centrasite -com:management:metrics:reference</name>
<description>
```

```
 Reference to a tModel containing all metrics about the Service
</description>
 <categoryBag>
   <keyedReference
     keyName="uddi-org:types:categorization"
     keyValue="categorization"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference
     keyName="uddi-org:types:checked"
     keyValue="checked"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference
     keyName="entityKeyValues"
     keyValue="tModelKey"
     tModelKey="uddi:uddi.org:categorization:entitykeyvalues"/>
 </categoryBag>
</tModel>
```

Metrics Types Taxonomy

The Metrics Types taxonomy is needed for classifying the metrics value set taxonomies:

```
<tModel tModelKey="uddi:centrasite.com:management:metrics:types">
 <name>CentraSite Metrics taxonomy </name>
<description>
Taxonomy holding all types of metrics known by CentraSite
</description>
 <categoryBag>
   <keyedReference
     keyName="uddi-org:types:categorization"
     keyValue="categorization"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference
     keyName="uddi-org:types:unchecked"
     keyValue="unchecked"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference
     keyName="entityKeyValues"
     keyValue="tModelKey"
     tModelKey="uddi:uddi.org:categorization:entitykeyvalues"/>
 </categoryBag>
</tModel>
```

Total Request Count Taxonomy

The following metrics tModel indicates the total number of requests:

```
<tModel tModelKey="uddi:centrasite.com:management:metrics:total.request.count">
 <name>Total Request Count</name>
 <description> Represents Metrics Total Request Count</description>
 <categoryBag>
   <keyedReference keyName="uddi-org:types:categorization"
     keyValue="categorization"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference keyName="uddi-org:types:unchecked"
     keyValue="unchecked"
     tModelKey="uddi:uddi.org:categorization:types"/>
   <keyedReference keyName="Metric Type"
     keyValue="Metrics Type"
     tModelKey="uddi:centrasite.com:management:metrics:types"/>
 </categoryBag>
</tModel>
```

Success Request Count Taxonomy

The following metrics tModel indicates the number of successful requests:

```
<tModel tModelKey="uddi:centrasite.com:management:metrics:success.request.count">
 <name>Success Request Count</name>
 <description>Represents Metrics Success  Request Count</description>
 <categoryBag>
   <keyedReference keyName="uddi-org:types:categorization"
     keyValue="categorization"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference keyName="uddi-org:types:unchecked"
     keyValue="unchecked"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference keyName="Metric Type"
     keyValue="Metrics Type"
     tModelKey="uddi:centrasite.com:management:metrics:types"/>
 </categoryBag>
</tModel>
```

Fault Request Count Taxonomy

The following metrics tModel indicates the number of faults:

```
<tModel tModelKey="uddi:centrasite.com:management:metrics:fault.request.count">
 <name>Fault Request Count</name>
 <description>Represents Metrics Fault Request Count</description>
 <categoryBag>
   <keyedReference keyName="uddi-org:types:categorization"
     keyValue="categorization"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference keyName="uddi-org:types:unchecked"
     keyValue="unchecked"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference keyName="Metric Type"
     keyValue="Metrics Type"
     tModelKey="uddi:centrasite.com:management:metrics:types"/>
 </categoryBag>
</tModel>
```

Average Response Time Taxonomy

The following metrics tModel indicates the average response time of requests:

```
<tModel tModelKey="uddi:centrasite.com:management:metrics:average.response.time">
 <name>Average Response Time</name>
 <description>Represents Metrics Average Response Time</description>
 <categoryBag>
   <keyedReference keyName="uddi-org:types:categorization"
     keyValue="categorization"
     tModelKey="uddi:uddi.org:categorization:types"/>
   <keyedReference keyName="uddi-org:types:unchecked"
     keyValue="unchecked"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference keyName="Metric Type"
     keyValue="Metrics Type"
     tModelKey="uddi:centrasite.com:management:metrics:types"/>
 </categoryBag>
</tModel>
```

Minimum Response Time Taxonomy

The following metrics tModel indicates the minimum response time of requests:

```
<tModel tModelKey="uddi:centrasite.com:management:metrics:minimum.response.time">
 <name>Minimum Response Time</name>
<description>Represents Metrics Minimum Response Time</description>
 <categoryBag>
   <keyedReference keyName="uddi-org:types:categorization"
     keyValue="categorization"
     tModelKey="uddi:uddi.org:categorization:types"/>
   <keyedReference keyName="uddi-org:types:unchecked"
     keyValue="unchecked"
     tModelKey="uddi:uddi.org:categorization:types"/>
   <keyedReference keyName="Metric Type"
     keyValue="Metrics Type"
     tModelKey="uddi:centrasite.com:management:metrics:types"/>
 </categoryBag>
</tModel>
```

Maximum Response Time Taxonomy

The following metrics tModel indicates the maximum response time of requests:

```
<tModel tModelKey="uddi:centrasite.com:management:metrics:maximum.response.time">
 <name>Maximum Response Time</name>
 <description>Represents Metrics Maximum Response Time</description>
 <categoryBag>
   <keyedReference keyName="uddi-org:types:categorization"
     keyValue="categorization"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference keyName="uddi-org:types:unchecked"
     keyValue="unchecked"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference keyName="Metric Type"
     keyValue="Metrics Type"
     tModelKey="uddi:centrasite.com:management:metrics:types"/>
 </categoryBag>
</tModel>
```

Availability Taxonomy

The following metrics tModel indicates the Virtual/Proxy service availability:

```
<tModel tModelKey="uddi:centrasite.com:management:metrics:availability">
 <name>Availability</name>
 <description>Represents Metrics Availability</description>
 <categoryBag>
   <keyedReference keyName="uddi-org:types:categorization"
```

```
     keyValue="categorization"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference keyName="uddi-org:types:unchecked"
     keyValue="unchecked"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference keyName="Metric Type"
     keyValue="Metrics Type"
     tModelKey="uddi:centrasite.com:management:metrics:types"/>
 </categoryBag>
</tModel>
```

Service Liveliness Taxonomy

The following metrics tModel indicates the uptime/downtime of Virtual/Proxy services:

```
<tModel tModelKey="uddi:centrasite.com:management:metrics:service.liveliness">
 <name>Service Liveliness</name>
 <description>Represents Metrics Service Liveliness</description>
 <categoryBag>
   <keyedReference keyName="uddi-org:types:categorization"
     keyValue="categorization"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference keyName="uddi-org:types:unchecked"
     keyValue="unchecked"
     tModelKey="uddi:uddi.org:categorization:types"/>

   <keyedReference keyName="Metric Type"
     keyValue="Metrics Type"
     tModelKey="uddi:centrasite.com:management:metrics:types"/>
 </categoryBag>
</tModel>
```

Representing Targets and Target Types

CentraSite stores Targets and Target Types as taxonomies. Thus, a deployment of a service to a target can be represented via a `uddi:keyedReference`.

Representing Status

Services that have a certain status are classified according to the LCM taxonomy.

Representing Version

Version information is represented by mapping the `jaxr:Slot` holding the version information. A `uddi:categoryBag` holding version information appears as follows:

```
<categoryBag>
 <keyedReference tModelKey="uddi:uddi.org:categorization:general_keywords"
                 keyName="Version" keyValue="1.0.0"/>
</categoryBag>
```

Mapping WS-PolicyAttachments

The mapping of WS-PolicyAttachments follows the W3C specification Web Service Policy Attachment version 1.5. This means that a tModel is created to represent the reusable policy expression.

Configuring the UDDI Environment

Configuration Properties

In CentraSite the UDDI registry reflects its behavior in terms of JAXR objects stored in the CentraSite Registry/Repository. This representation can be used to parameterize the behavior.

The behavior of the UDDI processing in CentraSite can be configured using global and local properties.

UDDI in a Multi-CAST Environment

CentraSite supports a multi-CAST (CentraSite Application Server Tier) UDDI registry environment. In this environment, multiple CASTs are running against a single CentraSite Registry/Repository. Each CAST comes with its own UDDI Registry web application. In addition, several pure JAXR-based clients interact with the Registry/ Repository.

In such an environment the Registry/Repository represents a single UDDI registry node, albeit multiple UDDI Registries are involved. Each UDDI Registry provides its own endpoints for the UDDI services. Due to the multi-CAST scenario, two different sets of properties are needed:

- Local properties, which specify the behavior of a single UDDI web application. Local properties can be used to change the behavior of the web applications independently.
- Global properties, which specify the global behavior of the UDDI registry. Global properties cannot be changed separately for each UDDI registry.

CAST Registration/Deregistration

You can register/deregister CASTs (and retrieve a list of CASTs) by executing the following commands in the command line interface `CentraSiteCommand.cmd` (Windows) or `CentraSiteCommand.sh` (UNIX) of CentraSite. The tool is located in <*CentraSiteInstallDir*> /utilities.

If you start this command line tool with no parameters, you receive a help text summarizing the required input parameters.

The parameters of the command are case-sensitive, so for example the parameter `-url` must be specified as shown and not as `-URL`.

Registering a CAST

Use the `list CAST` command to retrieve the list of available CASTs in CentraSite.

The syntax for the command is:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd list CAST [-url
<CENTRASITE-URL>] -user <USER-ID>-password <PASSWORD>
```

The input parameters are:

Parameter	Description
`-url`	**(Optional)** The URL of the CentraSite registry. Default value is `http:/localhost:53307`.
`-user`	The user ID of a registered CentraSite user. For example, a user who has the CentraSite Administrator role.
`-password`	The password for the registered CentraSite user identified by the parameter `-user`.

Example:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd list CAST -url
http://localhost:53305/CentraSite/CentraSite -user AdminUser -password
AdminPass
```

The response to this command could be:

```
Executing the command : list CAST
Successfully executed the command : list CAST
```

Registering a CAST

Use the `add CAST` command to register a CAST in CentraSite.

The syntax for the command is:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd add CAST [-url
<CENTRASITE-URL>] -user <USER-ID>-password <PASSWORD>
```

The input parameters are:

Parameter	Description
`-url`	**(Optional)** The URL of the CentraSite registry. Default value is `http:/localhost:53307`.

Parameter	Description
`-user`	The user ID of a registered CentraSite user. For example, a user who has the CentraSite Administrator role.
`-password`	The password for the registered CentraSite user identified by the parameter `-user`.

Example:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd add CAST -
url "http://localhost:53305/CentraSite/CentraSite" -user "AdminUser" -
password "AdminPass"
```

The response to this command could be:

```
Executing the command : add CAST
Successfully executed the command : add CAST
```

Deregistering a CAST

Use the `remove CAST` command to deregister a CAST from CentraSite.

The syntax for the command is:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd remove CAST [-
url <CENTRASITE-URL>] -user <USER-ID>-password <PASSWORD>
```

The input parameters are:

Parameter	Description
`-url`	**(Optional)** The URL of the CentraSite registry. Default value is `http:/localhost:53307`.
`-user`	The user ID of a registered CentraSite user. For example, a user who has the CentraSite Administrator role.
`-password`	The password for the registered CentraSite user identified by the parameter `-user`.

Example:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd remove CAST
-url http://localhost:53305/CentraSite/CentraSite -user AdminUser -
password AdminPass
```

The response to this command could be:

```
Executing the command : remove CAST
Successfully executed the command : remove CAST
```

Deployment Descriptors

In addition to the local and global properties that are stored in the registry, the web application needs a minimum of information to contact the CentraSite Registry/Repository. These parameters are:

- com.centrasite.uddi.store.db
- com.centrasite.uddi.store.dbUserId
- com.centrasite.uddi.store.dbUserPasswordHandle

The credentials are for the unauthenticated read access. Usually the guest account is used here.

Changing the User ID/Password of the Web Application Login Account

If you wish to change the password of the guest account, or if you wish to use another user account instead of the guest account, proceed as follows:

To change the user ID/password of the web application login account

1. In the file <*CentraSiteInstallDir* >\cast\cswebapps\UddiRegistry\WEB-INF\web.xml, specify the user ID and password in `com.centrasite.uddi.store.dbUserId` and `com.centrasite.uddi.store.dbUserPasswordHandle`.

   If your configuration includes more than one CAST, you must make this change on each CAST.
2. Update the corresponding user ID/password information that is stored in the CentraSite Registry Repository. For more information, see the *CentraSite Administrator's Guide* .
3. Restart CentraSite.

Setting Global and Local UDDI Properties

You can set the global and local UDDI properties by executing the following command in the command line interface `CentraSiteCommand.cmd` (Windows) or `CentraSiteCommand.sh` (UNIX) of CentraSite. The tool is located in <*CentraSiteInstallDir*> /utilities.

If you start this command line tool with no parameters, you receive a help text summarizing the required input parameters.

The parameters of the command are case-sensitive, so for example the parameter `-url` must be specified as shown and not as `-URL`.

Setting Global UDDI Properties

To set global properties

1. Create an XML configuration file that contains the following predefined UDDI properties. This file should be in Java XML properties format.

   For example:

   ```
   <?xml version="1.0" encoding="UTF-8" ?>
     <!DOCTYPE properties (View Source for full doctype...)>
     <properties version="1.0">
     <comment>Test UDDI Global Configuration XML</comment>
     <entry key="com.centrasite.uddi.UDDIOperatorName">CentraSite</entry>
     <entry key="com.centrasite.uddi.UDDIDefaultLanguage">en-US</entry>
     <entry key="com.centrasite.uddi.UDDIValueValidation">false</entry>
     <entry key="com.centrasite.uddi.UDDIKeyGeneratorChecks">true</entry>
     <entry key="com.centrasite.uddi.UDDIDiscoveryURLGeneration">false</entry>
     <entry key="com.centrasite.uddi.UDDIEncoding">utf-8</entry>
     <entry key="com.centrasite.uddi.UDDISubscriptionDuration">P1M</entry>
     <entry key="com.centrasite.uddi.UDDIAuthTokenExpiration">P1D</entry>
     <entry key="com.centrasite.uddi.UDDITransferTokenExpiration">P1D</entry>
     <entry key="com.centrasite.uddi.UDDIV2Inquiry">true</entry>
     <entry key="com.centrasite.uddi.UDDIV2Publish">true</entry>
     <entry key="com.centrasite.uddi.UDDISendEmptyWebServiceNotifications">
        false
     </entry>
     <entry key="com.centrasite.uddi.UDDISendEmptyEmailNotifications">
        true
     </entry>
     <entry key="com.centrasite.uddi.UDDIMinimalNotificationInterval">
        P0Y0M0DT0H0M30S
     </entry>
     <entry key="com.centrasite.uddi.UDDINumberOfRetries">3</entry>
     <entry key="com.centrasite.uddi.UDDIMaxSubscriptionThreads">5</entry>
     </properties>
   ```

   Descriptions of these properties are as follows:

Global Property	Description
UDDIOperatorName	Sets the `operator` attribute in UDDI V2 replies. The default value is `CentraSite`.
UDDIDefaultLanguage	Sets the default language. All valid language identifiers are allowed. The default value is “en-US”.
UDDIValueValidation	Activates or deactivates internal value validation. Valid values: "true" or "false" (default). For more information, see "Checked Value Set Validation" on page 126.

Global Property	Description
UDDIKeyGeneratorChecks	Activates or deactivates the enforcement of keyGenerator tModels for publisher assigned keys. Valid values: "true" (default) or "false".
UDDIDiscoveryURLGeneration	Activates or deactivates the generation of discoveryURL. Valid values: "true" or "false" (default).
UDDIEncoding	Specifies the XML encoding of the UDDI responses. Valid values: "utf-8" (default) or "utf-16".
UDDISubscriptionDuration	The duration of a registry subscription. You can specify a default subscription expiration period for each UDDI web application separately. The subscription duration is specified via an xs:duration instance. It specifies a duration in terms of years, months, days, hours, minutes and seconds. The default value is 1 month (xs:duration("P1M")). How a string holding an xs:duration instance is generated is specified in the XML Schema (Part 2) specification. For example, to specify a duration of 1 year, 2 months, 3 days, 10 hours and 30 minutes, specify: P1Y2M3DT10H30M. You can also indicate a duration of minus 120 days as: -P120D. Reduced precision and truncated representations of this format are allowed provided they conform to the following: If the number of years, months, days, hours, minutes, or seconds in any expression equals zero, the number and its corresponding designator *may* be omitted. However, at least one number and its designator *must* be present. The seconds part *may* have a decimal fraction. The designator 'T' shall be absent if all of the time items are absent. The designator 'P' must always be present.

Global Property	Description
	For example, P1347Y, P1347M and P1Y2MT2H are all allowed; P0Y1347M and P0Y1347M0D are allowed. P-1347M is not allowed although –P1347M is allowed. P1Y2MT is not allowed. **Note:** A UDDI subscription survives an application server restart.
UDDIAuthTokenExpiration	The duration of an authorization token. This duration is specified via an xs:duration instance. It specifies a duration in terms of years (either 0 or 1), months, days, hours, minutes and seconds. The default value is 1 day (xs:duration("P1D")).
UDDITransferTokenExpiration	The duration of a transfer token. This duration is specified via an xs:duration instance. It specifies a duration in terms of years (either 0 or 1), months, days, hours, minutes and seconds. The default value is 1 day (xs:duration("P1D")).
UDDIV2Inquiry	Enables UDDI V2 Inquiry support. Valid values: "true" (default) or "false".
UDDIV2Publish	Enables UDDI V2 Publish support. Valid values: "true" (default) or "false".
UDDISendEmptyWebServiceNotifications	Specifies whether to allow Web service notifications to be sent even when no changes have occurred. Valid values: "true" (default) or "false".
UDDISendEmptyEmailNotifications	Specifies whether to allow email notifications to be sent even when no changes have occurred. Valid values: "true" or "false" (default).

Global Property	Description
UDDIMinimalNotificationInterval	The minimum time interval at which to send notifications to subscribers. This duration is specified via an xs:duration instance. It specifies a duration in terms of years (either 0 or 1), months, days, hours, minutes and seconds. The default value is 30 seconds (xs:duration("P30S")). The minimal resolution is 10 seconds (as well as the minimal possible value).
UDDINumberOfRetries	The maximum number of times to try to send Web service notifications to subscribers. Valid values: 0 through 10. Default: 3. The value 0 turns off this property (i.e., allows for unlimited retries). **Note:** This property applies only to Web service notifications, not to email notifications.
UDDIMaxSubscriptionThreads	The maximum number of threads for subscriptions. Default: 5.

2. Use the `set UDDI` command to set the global UDDI properties in CentraSite.

   The syntax for the command is:

   ```
   C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd set UDDI
   [-url <CENTRASITE-URL>] -user <USER-ID>-password <PASSWORD>-file
   <CONFIG-FILE>
   ```

   The input parameters are:

Parameter	Description
`-url`	**(Optional)** The URL of the CentraSite registry. Default value is `http:/localhost:53307`.
`-user`	The user ID of a registered CentraSite user. For example, a user who has the CentraSite Administrator role.
`-password`	The password for the registered CentraSite user identified by the parameter `-user`.

Parameter	Description
`-file`	The absolute or relative path to the XML configuration file. If relative, the path should be relative to the location from where the command is executed.

Example:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd set UDDI -
url http://localhost:53305/CentraSite/CentraSite -user AdminUser -
password AdminPass -file config.xml
```

The response to this command could be:

```
Executing the command : set UDDI
Successfully executed the command : set UDDI
```

Setting Local UDDI Properties

Note: The UDDI representation of local properties is based on the general approach of mapping jaxr:Slots to UDDI.

To set local properties

1. Create an XML configuration file that contains the following predefined UDDI properties. This file should be in Java XML properties format.

   For example:

   ```
   <?xml version="1.0" encoding="UTF-8" ?>
     <!DOCTYPE properties (View Source for full doctype...)>
     <properties version="1.0">
     <comment>Test UDDI Local Configuration XML</comment>
     <entry key="com.centrasite.uddi.UDDIMaxResultSize">*</entry>
     <entry key="com.centrasite.uddi.UDDIMaxSearchKeys">*</entry>
     <entry key="com.centrasite.uddi.UDDIMaxSearchNames">*</entry>
     <entry key="com.centrasite.uddi.UDDIHTTPGetServicesUrl">
       http://localhost:53307/UddiRegistry
     </entry>
     <entry key="com.centrasite.uddi.UDDIResponseValidation">false</entry>
     <entry key="com.centrasite.uddi.UDDIRequestValidation">false</entry>
     </properties>
   ```

   Descriptions of these properties are as follows:

Local Property	Description
UDDIMaxResultSize	Specifies the maximum inquiry result size. Default: An unlimited size (denoted by *).
UDDIMaxSearchKeys	Specifies the maximum number of search keys returned by an inquiry. Default: An unlimited number (denoted by *).

Local Property	Description
UDDIMaxSearchNames	Specifies the maximum number of search names returned by an inquiry. Default: An unlimited number (denoted by *).
UDDIHTTPGetServicesUrl	Holds the URI for the HTTP Get calls to retrieve UDDI objects from the registry. Default value: http://localhost:53307/UddiRegistry.
UDDIResponseValidation	Enables schema validation on UDDI V2 and V3 responses. Valid values: "true" or "false" (default). **Note:** The appropriate schema file should be stored in the CentraSite repository, in the /projects/uddi folder.
UDDIRequestValidation	Enables schema validation on incoming UDDI V2 and V3 requests. Valid values: "true" or "false" (default). **Note:** The appropriate schema file should be stored in the CentraSite repository, in the /projects/uddi folder.

2. Use the `set UDDI` command to set your local UDDI properties in CentraSite.

   The syntax for the command is:

   ```
   C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd set UDDI
   [-url <CENTRASITE-URL>] -user <USER-ID>-password <PASSWORD>-file
   <CONFIG-FILE>
   ```

   The input parameters are:

Parameter	Description
`-url`	**(Optional)** The URL of the CentraSite registry. Default value is `http:/localhost:53307`.
`-user`	The user ID of a registered CentraSite user. For example, a user who has the CentraSite Administrator role.
`-password`	The password for the registered CentraSite user identified by the parameter `-user`.

Parameter	Description
`-file`	The absolute or relative path to the XML configuration file. If relative, the path should be relative to the location from where the command is executed.

Example:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd set UDDI -
url http://localhost:53305/CentraSite/CentraSite -user AdminUser -
password AdminPass -file config.xml
```

The response to this command could be:

```
Executing the command : set UDDI
Successfully executed the command : set UDDI
```

Getting Global and Local UDDI Properties

You can retrieve the global and local UDDI properties by executing the following command in the command line interface `CentraSiteCommand.cmd` (Windows) or `CentraSiteCommand.sh` (UNIX) of CentraSite. The tool is located in <*CentraSiteInstallDir*>/utilities.

If you start this command line tool with no parameters, you receive a help text summarizing the required input parameters.

The parameters of the command are case-sensitive, so for example the parameter `-url` must be specified as shown and not as `-URL`.

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd get UDDI [-url
<CENTRASITE-URL>] -user <USER-ID>-password <PASSWORD>
```

The input parameters are:

Parameter	Description
`-url`	**(Optional)** The URL of the CentraSite registry. Default value is `http:/localhost:53307`.
`-user`	The user ID of a registered CentraSite user. For example, a user who has the CentraSite Administrator role.
`-password`	The password for the registered CentraSite user identified by the parameter `-user`.

Example:

```
C:\SoftwareAG\CentraSite\utilities>CentraSiteCommand.cmd get UDDI -url
http://localhost:53305/CentraSite/CentraSite -user AdminUser -password
AdminPass
```

The response to this command could be:

```
Executing the command : get UDDI
Successfully executed the command : get UDDI
```

Schema Validation of UDDI Requests

All incoming UDDI V3 requests are validated against the XML schemas for UDDI requests, as defined in the OASIS UDDI specification. Invalid requests are rejected. The schema files are stored in the CentraSite repository in the folder /projects/uddi.

The validation can be activated or deactivated by the local configuration properties `Request Validation` and `Response Validation`. For details, see "Setting Global and Local UDDI Properties" on page 118.

Checked Value Set Validation

CentraSite offers the internal validation of checked value sets. If a `keyedReference` object is published that points to a checked value set, CentraSite checks if the `keyValue` belongs to the value set. UDDI 3.0.2 specifies checked value sets specified by value set taxonomies and value sets that have a validation algorithm. All value sets are represented in UDDI by a tModel. The validation of `keyedReference` objects can be switched off. For this purpose, the global configuration property `Internal Value Set Validation` can be used. To set this property, see "Setting Global and Local UDDI Properties" on page 118.

If the property is set to `false`, no checking is performed but `keyedReference` objects pointing to an existing taxonomy are still mapped to a `jaxr:internal` classification. Invalid references are mapped to external classifications. This means that even a `keyedReference` object with an invalid value that points to a classificationScheme with a concept taxonomy is not rejected but mapped to an external classification. If the property is set to `yes`, `keyedReference` objects pointing to checked value sets are checked. Invalid `keyedReference` objects are rejected with the error `E_invalidValue`.

Predefined Value Sets

CentraSite supports the following value sets from the UDDI specification:

- Type category system
- Relationships category system
- Entity Key Values category system
- NAICS 1997 Release
- NAICS 2002 Release
- UNSPSC Version 7.3

- ISO 3166 Geographic Code System
- UDDI v2 OwningBusiness category system
- UDDI v2 IsReplacedBy identifier system
- UDDI Entity Key Values category System
- UDDI `Derived From` category system
- UDDI Nodes category system
- General Keyword category system
- Postal Address Structure

Predefined tModels

The following is a list of the predefined tModels in CentraSite:

```
AssociationType
CentraSite
CentraSiteFilterType
ClassificationGroup
ContentType
Databases
Interstage Business Process Manager
Object
ObjectType
Origin
PhoneType
PostalAddressAttributes
Products
RepositoryObjectType
URLType
UseType
crossvision Application Composer
crossvision Information Integrator
crossvision Legacy Integrator - ApplinX
crossvision Legacy Integrator - EntireX
crossvision Service Orchestrator
dnb-com:D-U-N-S
http://schemas.xmlsoap.org/ws/2003/03/localpolicyreference
http://schemas.xmlsoap.org/ws/2003/03/policytypes
http://schemas.xmlsoap.org/ws/2003/03/remotepolicyreference
node-centrasite-com:keyGenerator
ntis-gov:naics:1997
ntis-gov:naics:2002
ntis-gov:sic:1987
thomasregister-com:supplierID
ubr-uddi-org:iso-ch:3166-2003
ubr-uddi-org:iso-ch:6523-1998:icd
ubr-uddi-org:postalAddress
uddi-org: protocol:keyGenerator
uddi-org:UTS-10
uddi-org:andAllKeys
uddi-org:approximateMatch:SQL99
uddi-org:binarySort
uddi-org:bindingSubset
uddi-org:caseInsensitiveMatch
```

```
uddi-org:caseInsensitiveSort
uddi-org:caseSensitiveMatch
uddi-org:caseSensitiveSort
uddi-org:categorization:keyGenerator
uddi-org:combineCategoryBags
uddi-org:derivedFrom
uddi-org:diacriticsInsensitiveMatch
uddi-org:diacriticsSensitiveMatch
uddi-org:entityKeyValues
uddi-org:exactMatch
uddi-org:fax
uddi-org:ftp
uddi-org:general_keywords
uddi-org:homepage
uddi-org:hostingRedirector
uddi-org:http
uddi-org:inquiry
uddi-org:inquiry_v2
uddi-org:inquiry_v3
uddi-org:isReplacedBy
uddi-org:keyGenerator
uddi-org:mutualAuthenticatedSSL3
uddi-org:node_custody_transfer_v3
uddi-org:nodes
uddi-org:orAllKeys
uddi-org:orLikeKeys
uddi-org:ownership_transfer_v3
uddi-org:owningBusiness_v3
uddi-org:protocol:http
uddi-org:protocol:soap
uddi-org:publication
uddi-org:publication_v2
uddi-org:publication_v3
uddi-org:relationships
uddi-org:replication_v3
uddi-org:security_v3
uddi-org:serverAuthenticatedSSL3
uddi-org:serviceSubset
uddi-org:signaturePresent
uddi-org:smtp
uddi-org:sortByDateAsc
uddi-org:sortByDateDesc
uddi-org:sortByNameAsc
uddi-org:sortByNameDesc
uddi-org:sortorder:keyGenerator
uddi-org:suppressProjectedServices
uddi-org:taxonomy
uddi-org:taxonomy_v2
uddi-org:telephone
uddi-org:transport:keyGenerator
uddi-org:types
uddi-org:v3_policy
uddi-org:validatedBy
uddi-org:valueSetCaching_v3
uddi-org:valueSetValidation_v3
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
uddi.org:bpel:types
```

```
uddi:ebxml.org:collaborationprotocolagreement:v1.0:template
uddi:ebxml.org:collaborationprotocolagreement:v2.0:template
uddi:ebxml.org:collaborationprotocolprofile:v1.0
uddi:ebxml.org:collaborationprotocolprofile:v2.0
uddi:ebxml.org:messageservice:v1.0
uddi:ebxml.org:messageservice:v2.0
uddi:oasis-open.org:wsrp:v1_bindings
uddi:oasis-open.org:wsrp:service_type
uddi:oasis-open.org:wsrp:v1_service_description_porttype
uddi:oasis-open.org:wsrp:v1_markup_porttype
uddi:oasis-open.org:wsrp:v1_registration_porttype
uddi:oasis-open.org:wsrp:v1_portlet_management_porttype
uddi:oasis-open.org:wsrp:v1_service_description_binding_soap
uddi:oasis-open.org:wsrp:v1_markup_binding_soap
uddi:oasis-open.org:wsrp:v1_registration_binding_soap
uddi:oasis-open.org:wsrp:v1_portlet_management_binding_soap
uddi:uddi.org:bpel:wsdlporttypereference
uddi:uddi.org:propertyset
uddi:uddi.org:update_entities_v2
uddi:uddi.org:update_entities_v3
uddi:w3.org:ws-policy:v1.5:attachment:policytypes
uddi:w3.org:ws-policy:v1.5:attachment:remotepolicyreference
uddi:w3.org:ws-policy:v1.5:attachment:localpolicyreference
uddi:untmg.org:businessprocessspecificationschema:v1.10
unspsc-org:unspsc
unspsc-org:unspsc:3-1
unspsc-org:unspsc:v6.0501
```

UDDI V3 APIs

Overview

The CentraSite Registry/Repository supports the Java API for XML Registries (JAXR). It also supports UDDI V3- and V2. These APIs enable you to interact with the CentraSite Registry/Repository directly from UDDI-compliant browsers and integration development environment (IDE) tools. For more information about IDE tools, see "Using Third-Party IDE Tools with CentraSite " on page 135.

CentraSite provides Javadocs that you can use to create UDDI V3 clients. The Javadocs provide the interfaces you need for implementing the Publish, Inquiry, Security and Taxonomy APIs in your clients.

`<CentraSite_installation_root>\Documentation\en\jd\uddiv3ClientAPI`

CentraSite supports the following APIs.

Use this API...	To enable the client to...
Publish	Execute any UDDI publishing API call. For example, you can publish services to CentraSite and publish proxy endpoints for services that already exist in CentraSite. You can publish the following UDDI objects: Organization, Service, ServiceBinding and tModel.

Use this API...	To enable the client to...
Inquiry	Interrogate CentraSite to retrieve service information. When an active run-time policy’s virtual service executes, the Inquiry API will pull the virtual service's information from CentraSite. You can publish the following UDDI objects: Organization, Service, ServiceBinding and tModel.
Security	Execute UDDI security API calls, using authorization tokens.
Taxonomy	Fetch taxonomies and their immediate children. The taxonomies are represented in the tree structure. The Taxonomy API is a custom API.

Classes and Interfaces

The major classes and interfaces available in the Javadocs are described below.

RegistryService

`RegistryService` is the core interface to communicate with the UDDI Registry using the UDDI V3 API. This interface contains utility methods to get service stubs for the Publish, Inquiry, Security and Taxonomy APIs. The UDDI operations are performed using their respective service stubs. It also contains a method to connect to the CentraSite Registry/Repository, using authentication tokens.

RegistryConfiguration

`RegistryConfiguration` is a bean class that is used to connect to the CentraSite Registry/Repository, based on the Registry/Repository’s configuration details, such as its host, port and URLs. The URLs include the Security URL, Inquiry URL, Publish URL and Taxonomy URL. This class also contains the user credentials of the Registry/Repository.

RegistryFramework

`RegistryFramework` is a helper interface that can be used to get attribute values, relationships and documents. This class contains the helper method `getServiceModifiedDate`, which uses `get_operationalInfo` to get the modified date of the service.

Note: Unlike the UDDI specification, serviceBinding is a contained element of a businessService entity in CentraSite. Thus, when a service's bindingTemplate entity is updated, then the service will be updated as well. This means that when you use the helper method `getServiceModifiedDate`, it will return the same modification time for both the `modified` and `modifiedIncludingChildren` attributes.

RegistryAgent

`RegistryAgent` is an interface that enables a policy enforcement point to query the virtual services in the CentraSite Registry/Repository, and to publish run-time performance metrics to the CentraSite Registry/Repository.

This interface contains the following helper methods:

- `findVirtualServices`, which finds the virtual services that are deployed to the runtime target
- `getVirtualServiceWSDLURL`, which returns the WSDL URL for the specified virtual service
- `saveMetrics`, which saves the run-time performance metrics for the virtual services

UDDI_Security_SoapService

`UDDI_Security_SoapService` is an interface that can be used for all UDDI Security operations. This interface contains the methods `get_authToken` and `discard_authToken`. An instance of this interface can be obtained from `RegistryService`.

UDDI_Inquiry_SoapService

`UDDI_Inquiry_SoapService` is an interface that contains methods for all Inquiry operations. An instance of this interface can be obtained from `RegistryService`.

UDDI_Publication_SoapService

`UDDI_Publication_SoapService` is an interface that contains methods for all UDDI Publish operations. An instance of this interface can be obtained from `RegistryService`.

UDDI_Taxonomy_SoapService

`UDDI_Taxonomy_SoapService` is an interface that uses the method `get_conceptDetail` to fetch taxonomies and their immediate children. The taxonomies are represented in the tree structure. An instance of this interface can be obtained from `RegistryService`.

CentraSiteBusinessService

`CentraSiteBusinessService` is a wrapper class for the `BusinessService` class. This class contains all methods contained in `BusinessService`, as well as these additional methods:

- `getAttachedPolicyDocURL`, which returns the attached policy associated with the service.

- `getAttachedPolicyTModelKey`, which returns the attached policy tModel key associated with the service.
- `getAttributes`, which returns the attributes associated with the service in a Map.
- `getDocuments`, which returns the documents associated with the service in a Map.
- `getRelatedObjectKey`, which returns the UDDI key of the object that has the specified relationship.

Examples

Getting the Value of an Attribute

The following example shows how to get the value of an attribute named `Life Cycle Status`.

```
//Creating configuration object with host, port
//and user credentials of the registry
RegistryConfiguration regConfig =
  new RegistryConfiguration("localhost", "53307",
  "DefaultUser", "PwdFor_CS21");

//Creating registry service instance using the RegistryConfiguration
RegistryService regService =
  RegistryService.Factory.newInstance(regConfig);

//connection is made (get_authToken will be issued to registry)
regService.connect();

//Inquiring the registry for the service using find_service call

UDDI_Inquiry_SoapService inquirySoapService =
  regService.getInquirySoapService();
FindService findService = new FindService();
Name name = new Name();
name.setValue("UDDI Security Service");
findService.setName(new Name[] {name});
findService.setAuthInfo(regService.getAuthToken());
System.out.println("Name....."+ findService);
ServiceList serviceList = inquirySoapService.find_service(findService);
ServiceInfos serviceInfos = serviceList.getServiceInfos();

//Getting the service Key for the first service
ServiceInfo serviceInfo = serviceInfos.getServiceInfo(0);
String serviceKey = serviceInfo.getServiceKey();

//Getting the service detail
GetServiceDetail getServiceDetail = new GetServiceDetail();
getServiceDetail.setServiceKey(new String[] {serviceKey});
getServiceDetail.setAuthInfo(regService.getAuthToken());
ServiceDetail serviceDetail =
  inquirySoapService.get_serviceDetail(getServiceDetail);
BusinessService businessService =
  serviceDetail.getBusinessService(0);

//Creating instance of CentraSiteBusinessService
CentraSiteBusinessService csBusinessService = new
CentraSiteBusinessService(businessService);
```

```
//Getting the value for the attribute "Life Cycle Status"
String attributeValue =
  csBusinessService.getAttributeValue("Life Cycle Status");
```

Getting the Proxy Services for a Specified Target

The following example shows how to get the proxy services for a specified target.

```
//Creating configuration object with host, port
//and user credentials of the registry
RegistryConfiguration regConfig =
  new RegistryConfiguration("localhost",
  "53307", "DefaultUser", "PwdFor_CS21");

//Creating registry service instance using the RegistryConfiguration
RegistryService regService =
  RegistryService.Factory.newInstance(regConfig);

//connection is made (get_authToken will be issued to registry)
regService.connect();

//Getting the RegistryAgent instance using RegistryService
RegistryAgent registryAgent = regService.getRegistryAgent();

//Getting the ServiceInfos which will contain
//a list of the services deployed in the "Actional" target
ServiceInfos proxyServices =
  registryAgent.findProxyServices("Actional");
```

Inquiring about a Business Service

The following example shows how to fetch the details of a business service, using the UDDI Inquiry API.

```
//RegistryConfiguration containing the host, port, userId and
//password to connect to registry
RegistryConfiguration regConfig =
  new RegistryConfiguration("hostName", "port", "userId", "password");

//Creating the RegistryService using RegistryConfiguration
RegistryService regService =
  RegistryService.Factory.newInstance(regConfig);

//connecting to registry. This method will fetch the AuthToken
//using get_authTokenAPI
regService.connect();

//Inquiring the registry for the service using find_service call
UDDI_Inquiry_SoapService inquirySoapService =
  regService.getInquirySoapService();

//Constructing the find_service inquiry call
FindService findService = new FindService();
Name name = new Name();
name.setValue("UDDI Inquiry Service");
findService.setName(new Name[] {name});

//Issuing find_service inquiry call to
//CentraSite registry using UDDI_Inquiry_SoapService
ServiceList serviceList = inquirySoapService.find_service(findService);
ServiceInfos serviceInfos = serviceList.getServiceInfos();
```

```
//Getting the service Key for the first service
ServiceInfo serviceInfo = serviceInfos.getServiceInfo(0);
String serviceKey = serviceInfo.getServiceKey();

//Getting the service detail
GetServiceDetail getServiceDetail = new GetServiceDetail();
getServiceDetail.setServiceKey(new String[] {serviceKey});
getServiceDetail.setAuthInfo(regService.getAuthToken());
ServiceDetail serviceDetail =
  inquirySoapService.get_serviceDetail(getServiceDetail);
BusinessService businessService =
  serviceDetail.getBusinessService(0);
System.out.println("Fetched Service Name : " +
  businessService.getName()[0].getValue());
```

Publishing a Business Service

The following example shows how to publish a business service, using the UDDI Publish API.

```
//RegistryConfiguration containing the host, port,
//userId and password to connect to registry
RegistryConfiguration regConfig =
  new RegistryConfiguration("hostName", "port", "userId", "password");

//Creating the RegistryService using RegistryConfiguration
RegistryService regService =
  RegistryService.Factory.newInstance(regConfig);

//connecting to registry. This method will fetch the
//AuthToken using get_authTokenAPI
regService.connect();

//Getting the UDDI_Publication_SoapService to publish the
//sample business service
UDDI_Publication_SoapService publishSoapService =
  regService.getPublishSoapService();

//Constructing the save service call for sample business service
SaveService saveService = new SaveService();
BusinessService businessService = new BusinessService();
Name name = new Name();
name.setValue("Sample Business Service");
businessService.setName(new Name[] {name});

//Setting the auth token using the registry service
saveService.setAuthInfo(regService.getAuthToken());
saveService.setBusinessService(new BusinessService[] {businessService});

//Saving the business service using UDDI_Publication_SoapService
publishSoapService.save_service(saveService);
```

Fetching Taxonomies

The following example shows how to fetch taxonomies, using the Taxonomy API.

```
//RegistryConfiguration containing the host, port,
//userId and password to connect to registry
RegistryConfiguration regConfig =
  new RegistryConfiguration("hostName", "port", "userId", "password");
```

```
//Creating the RegistryService using RegistryConfiguration
RegistryService regService =
  RegistryService.Factory.newInstance(regConfig);

//connecting to registry. This method will fetch the
//AuthToken using get_authTokenAPI
regService.connect();

//Getting the taxonomy soap service which is used fetch the taxonomies
UDDI_Taxonomy_SoapService taxonomySoapService =
  regService.getTaxonomySoapService();

//Constructing the get_conceptDetail request
GetConceptDetail getConceptDetail = new GetConceptDetail();

//Fetching the NAICS taxonomy
getConceptDetail.setConceptKey(new String[]
  {"uddi:uddi.org:ubr:categorization:naics:1997"});

//Using UDDI_Taxonomy_SoapService we are fetching the
//taxonomies from CentraSite registry
ConceptDetail conceptDetail =
  taxonomySoapService.get_conceptDetail(getConceptDetail);
```

Using Third-Party IDE Tools with CentraSite

Overview

An Integrated Development Environment (IDE) tool for Web services is a user interface provided by any vendor that enables you to publish (submit data) and inquire (search data) in any UDDI registry. CentraSite supports any IDE tool that complies with WSDL and UDDIV3 or UDDIV2 inquiry and publish semantics. Using IDE tools with CentraSite, you can publish, inquire and delete Web services.

Supported IDE Tools

The following IDE tools can be used with CentraSite version 9.8.

- WTP Eclipse 1.5.2 plug-in
- IBM Rational Application Developer 6.0
- Parasoft JTest 7.5
- PushToTest
- UDDI4J
- RUDDI

Specifying the Inquiry, Publish and Security URLs

When using any IDE tool, you need to obtain a user account in CentraSite, and also provide an inquiry URL for inquiring a Web service and a publish URL for publishing a

Web service. You also need to provide the security URL for those tools that require it (for example, ALSB uses the security URL to get the AuthToken).

The UDDI Publish, Inquiry and Security services are hosted at the following URLs on the CentraSite host machine:

Inquiry URL:

`http://<hostName>:<port>/UddiRegistry/inquiry`

Publish URL:

`http://<hostName>:<port>/UddiRegistry/publish`

Security URL:

`http://<hostName>:<port>/UddiRegistry/security`

where *<hostName>* is the host name or IP address of the machine on which CentraSite is installed and *<port>* is the port on which CentraSite is listening for http requests.

Note: The `save_binding` call in UDDI sends the access point but does not send the WSDL URL. Therefore, the WSDL URL shows only the access point.

WTP Eclipse 1.5.2 Plug-In

You can search for a business and publish a service from that business, using the WTP Eclipse client.

To download the plug-in and access the Web Services Explorer

1. Download the Eclipse Web Tools version 1.5.2 from http://download.eclipse.org/webtools/downloads/. This version contains Eclipse 3.2.1.
2. Run your virus scan product to ensure the Eclipse Web Services Explorer opens properly.
3. On the Eclipse SDK screen, click **Window > Open Perspective**.
4. On the **Open Perspective** dialog box, select **J2EE** and then click **OK**.
5. Click **Run > Launch the Web Services Explorer**.

   The **Web Services Explorer** portlet displays.
6. In the Navigator pane, click **UDDI Main**.
7. On the Open Registry screen, type the following URL in the Inquiry URL box:

   `http://<hostName>:<port>/UddiRegistry/inquiry`

   where *<hostName>* is the host name or IP address of the machine on which CentraSite is installed and *<port>* is the port on which CentraSite is listening for http requests.
8. Click **Go**.

If the registry is active, Eclipse displays its details. On this page, you can find and publish services, businesses and service interfaces (tModels).

9. Click **Go**.

IBM Rational Application Developer 6.0

Connecting to CentraSite

For more information about the IBM Rational Application Developer 6.0, see the IBM developerWorks website.

To connect to CentraSite

1. From the **Window** menu of the tool, click **Open Perspective > J2EE**.
2. Click **Run > Launch the Web Services Explorer**.
3. In the navigation window, click **UDDI Main**.
4. In the **Registry Name** box, type `CentraSite`.
5. In the **Inquiry URL** box, type `http://<host>:<port>/UDDIRregistry/inquiry`, where *<host>* and *<port>* reflect the target CentraSite registry.

Publishing Entities

To publish entities

1. In the Publish URL box, type `http://<host>:<port>/UDDIRegistry/publish`, where *<host>* and *<port>* reflect the target CentraSite registry.
2. Supply your user name and password.
3. Identify the registry to which you want to publish the entity.
4. Provide entity details.

   When you attempt to find services and publish a new service, double authentication is required. But when you attempt to publish a service directly, authentication is involved once.

UDDI Extensions

Various extensions to the UDDI standard have been published by the standards bodies OASIS and W3C. The extensions described below are implemented in CentraSite's UDDI environment.

Using WSDL in a UDDI Registry

Since a UDDI registry houses information about web services and their providers, it is essential that the information contained in a web service's WSDL document is accurately mapped to the UDDI data model. This means that subsequent search operations to discover a registered web service are possible, based on the information that is mapped from the WSDL.

How CentraSite performs WSDL-to-UDDI mappings is based on the recommendations that OASIS has published. The technical description of the OASIS recommendation for WSDL-to-UDDI mapping can be found on the OASIS website.

Using WS-PolicyAttachment

An XML-based expression grammar for policies is described in the Web Services Policy Framework (WS-Policy) specification, published by the W3C. The specification also describes how a policy can be associated with a registry object.

The CentraSite UDDI registry supports WS-PolicyAttachment version 1.2 and 1.5. Policy attachments can be either WSDL-based or UDDI-based. Currently, CentraSite supports UDDI-based policy attachments. With UDDI-based policy attachments, the policies are modeled in the UDDI registry using UDDI elements.

Version 1.2 Support

The CentraSite WS-PolicyAttachment support for version 1.2 covers the following aspects of the WS-PolicyAttachment specification:

Supported Policy Subjects

CentraSite supports the UDDI-based policy attachments for the following policy subjects:

- Service Provider Policy Subject
- Service Policy Subject
- Endpoint Policy Subject

Referencing Remote Policy Expressions

An example for a remote policy reference is shown by the following uddi:businessService that is taken from the WS-PolicyAttachment specification:

```
<businessService serviceKey="…" >
 <name>…</name>
 <description>…</description>
 <bindingTemplates>…</bindingTemplates>
 <categoryBag>
   <keyedReference
      keyName="Policy Expression for example's Web services"
      keyValue="http://www.example.com/myservice/policy"
      tModelKey="uuid:a27078e4-fd38-320a-806f-6749e84f8005" />
 </categoryBag>
</businessService>
```

The uddi:businessService is attached to a WS-Policy that accessible through the URL http://www.example.com/myservice/policy. The uddi:keyedReference represents the attachment. It is referencing the remote policy reference category system via its uddi:tModelKey and its value holds the URI of the policy document.

Registering Reusable Policy Expressions

A reusable policy expression is represented by a dedicated uddi:tModel in the UDDI registry. The following uddi:tModel shows an example:

```
<tModel tModelKey="uuid:04cfa…">
 <name>…</name>
<description xml:lang="EN">
   Policy Expression for example's Web services
 </description>
 <overviewDoc>
   <description xml:lang="EN">WS-Policy Expression</description>
   <overviewURL>http://www.example.com/myservice/policy</overviewURL>
 </overviewDoc>
 <categoryBag>
   <keyedReference
      keyName="Reusable policy Expression"
      keyValue="policy"
      tModelKey="uuid:fa1d77dc-edf0-3a84-a99a-5972e434e993" />
   <keyedReference
      keyName="Policy Expression for example's Web services"
      keyValue="http://www.example.com/myservice/policy"
      tModelKey="uuid:a27078e4-fd38-320a-806f-6749e84f8005" />
 </categoryBag>
</tModel>
```

The uddi:tModel comes with two uddi:keyedReferences. The first uddi:keyedReference specifies the uddi:tModel to represent a reusable policy expression. The second one points to the document holding the policy expression. An example that shows the attachment of a reusable policy expression is given by the following uddi:businessService:

```
<businessService serviceKey="…" >
 <name>…</name>
 <description>…</description>
 <bindingTemplates>…</bindingTemplates>
 <categoryBag>
   <keyedReference
      keyName="Policy Expression for example's Web services"
      keyValue="uuid:04cfa…"
      tModelKey="uuid:a27f7d45-ec90-31f7-a655-efe91433527c" />
 </categoryBag>
</businessService>
```

The uddi:businessService holds a keyedReference pointing to the uddi:tModel holding the reusable policy expression. The uddi:tModelKey of the uddi:keyedReference points to the local policy reference uddi:tModel.

Registering Policies in UDDI Version 3

CentraSite supports UDDI-based policy attachments for UDDI version 2 and 3.

tModels to Support UDDI-Based WS-PolicyAttachments

CentraSite provides the tModels necessary to support UDDI-based WS-PolicyAttachments.

Remote Policy Reference Category System

This tModel is used to attach a policy to a UDDI entity by referencing the policy's URI.

```
<tModel tModelKey="uddi:schemas.xmlsoap.org:remotepolicyreference:2003_03" >
 <name>http://schemas.xmlsoap.org/ws/2003/03/remotepolicyreference</name>
 <description xml:lang="EN">
   Category system used for UDDI entities to point to an external
   WS-PolicyAttachment Policy Expression that describes their
   characteristics. See WS-PolicyAttachment specification for further details.
 </description>
 <categoryBag>
   <keyedReference
      keyName="uddi-org:types:categorization"
      keyValue="categorization"
      tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4" />
 </categoryBag>
</tModel>
```

WS-Policy Types Category System

This tModel is used to categorize tModels as representing Policy Expressions. There is only one valid value, namely `policy`, that indicates this very fact. It is *recommended* that tModels categorized as representing Policy Expressions reference no more and no less than this very Policy Expression using the Remote Policy Reference category system.

```
<tModel tModelKey=" uddi:schemas.xmlsoap.org:policytypes:2003_03" >
 <name>http://schemas.xmlsoap.org/ws/2003/03/policytypes</name>
 <description xml:lang="EN">
   WS-Policy Types category system used for UDDI tModels to characterize them
   as WS-Policy - based Policy Expressions.
 </description>
 <categoryBag>
   <keyedReference
      keyName="uddi-org:types:categorization"
      keyValue="categorization"
      tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4" />
 </categoryBag>
</tModel>
```

Local Policy Reference Category System

This tModel is used to attach a Policy Expression to a UDDI entity by referencing the UDDI entity that represents this Policy Expression. The Local Policy Reference category system is based on tModelKeys. It is expected that referenced tModels are registered with the same UDDI registry and are categorized as representing Policy Expressions using the WS-Policy Types category system.

```
UDDI Key (V3): uddi:schemas.xmlsoap.org:remotepolicyreference:2003_03
UDDI V1,V2 format key: uuid:a27f7d45-ec90-31f7-a655-efe91433527c
Categorization: categorization
Checked: Yes
<tModel tModelKey="uddi:schemas.xmlsoap.org:localpolicyreference:2003_03" >
 <name>http://schemas.xmlsoap.org/ws/2003/03/localpolicyreference</name>
 <description xml:lang="en">
   Category system used for UDDI entities to point to a WS-Policy
   Policy Expression tModel that describes their characteristics.
   See WS-PolicyAttachment specification for further details.
 </description>
 <categoryBag>
   <keyedReference
      keyName="uddi-org:types:categorization"
      keyValue="categorization"
      tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62aB4" />
   <keyedReference
      keyName="uddi-org:entityKeyValues"
      keyValue="tModelKey"
      tModelKey="uuid:916b87bf-0756-3919-8eae-97dfa325e5a4" />
 </categoryBag>
</tModel>
```

Version 1.5 Support

The CentraSite WS-PolicyAttachment support for version 1.5 covers the same aspects as for version 1.2. Additionally, CentraSite provides the following tModels necessary to support UDDI-based WS-PolicyAttachments:

- uddi:w3.org:ws-policy:v1.5:attachment:localpolicyreference
- uddi:w3.org:ws-policy:v1.5:attachment:policytypes
- uddi:w3.org:ws-policy:v1.5:attachment:remotepolicyreference

Extending UDDI Publisher API Set to Enable Physical Deletion of tModels

For the physical deletion of un-referenced tModels, CentraSite extends the UDDI Publication API set. Appendix H of the UDDI V3 Specification describes how UDDI can be extended. For removing a tModel from a registry, CentraSite introduces the request `purge_tModel`. This extension is only supported for UDDI version 3.

Arguments

The `purge_tModel` request has the same arguments as the `delete_tModel` V3 request:

- `authInfo`: This optional argument is an element that contains an authentication token.
- `tModelKey`: One or more required uddiKey values that represent specific instances of known tModel data.

Behavior

The request removes a hidden tModel from the registry that is not referenced by any other UDDI object.

Returns

On successful completion an empty message is returned.

Caveats

In the case of an error, a disposition report will be returned within a SOAP fault. In addition to the errors which are common to all API calls, the following errors are relevant here:

- `E_invalidKeyPassed`: Signifies that one of the uddiKey values passed did not match any known tModelKey values, or multiple instances of the same tModelKey values were passed. The error is also returned if the referenced tModel is not hidden, or it is still referenced. This means there are four conditions for this error. The different conditions should be reflected in the `errInfo` element:
  - The specified tModel cannot be found
  - Multiple references to the same tModel
  - The specified tModel is not hidden
  - The specified tModel is still referenced
- `E_userMismatch`: Signifies that one or more of the tModelKey values passed refers to data that is not owned by the individual publisher who is represented by the authentication token.

4 Pluggable Architecture

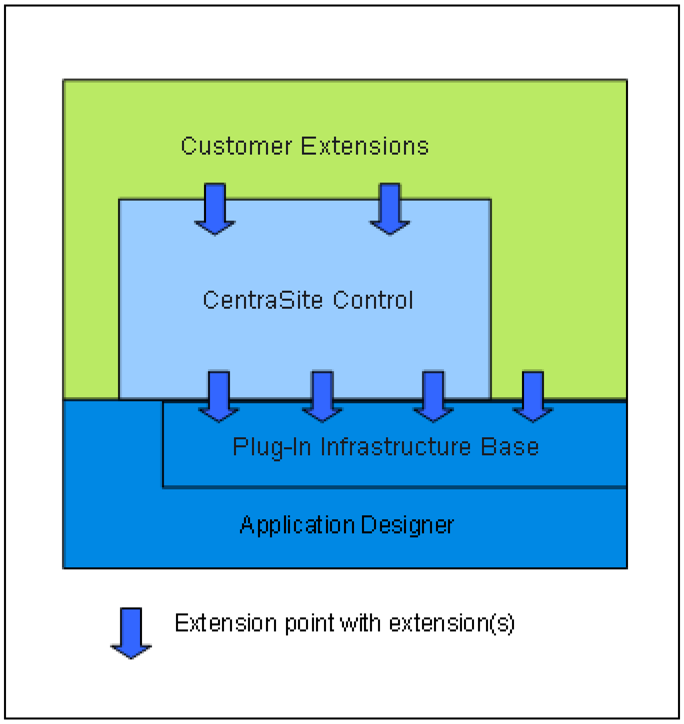
- Introduction to CentraSite Control Pluggable Architecture 144
- Customizing the Welcome Page 145
- Customizing Content Pages 167
- Setting the Preferred Plug-In and Order of Plug-Ins 186
- Installing and Uninstalling Plug-Ins 186
- Special and Advanced Topics 189
- Javadoc Documentation of the APIs 192
- Step-by-Step Guide 192

Introduction to CentraSite Control Pluggable Architecture

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 pluggable architecture is illustrated in the following diagram:


Customer Extensions
CentraSite Control
Plug-In Infrastructure Base
Application Designer
Extension point with extension(s)

The plug-in infrastructure is inspired by Eclipse, which 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 / 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.

The available extension points are described in "Extension Points" on page 167.

Customizing the Welcome Page

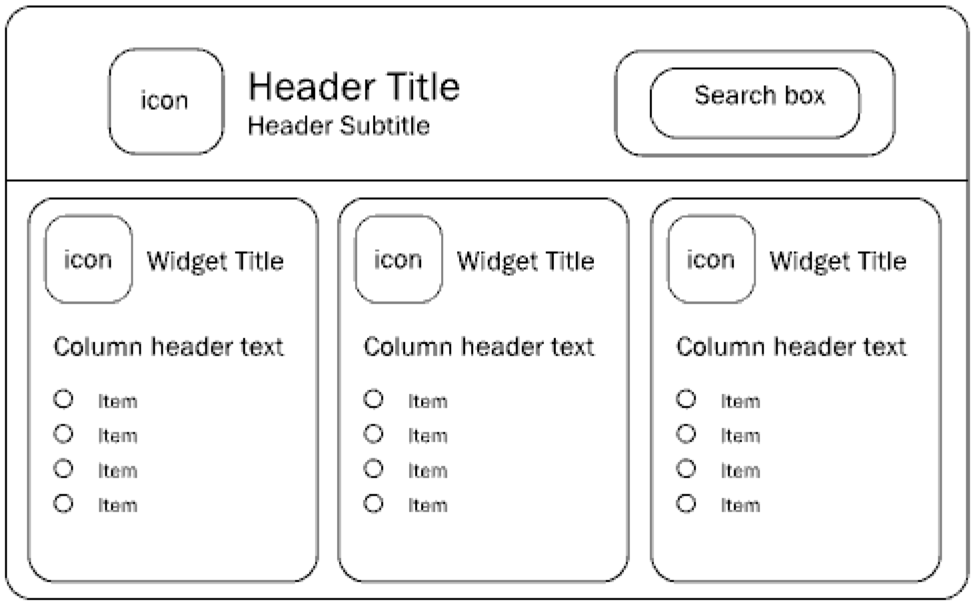
The Welcome page that you see when you start CentraSite Control can be customized to suit your own requirements. You can change aspects such as icons used, colors, text, fonts and layouts. You can also define links that will take you straight to the pages of CentraSite Control that you use the most, and links to external web sites.

Introduction

The standard Welcome page gives you quick links to the pages of CentraSite Control that you will probably 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. In the Welcome page you can specify the language you wish to use for your further work with CentraSite Control, and you can specify the date format to be used in the various displays.

A search box allows you perform a keyword search for registry assets and objects whose name or description contains the given keyword.

The Welcome page has the following schematic layout:



The header section at the top contains a title text and a subtitle text. You can change the texts, the fonts and colors used to display the texts. An icon can be displayed adjacent to the title and subtitle. A search box is displayed by default, which allows you to perform a keyword search for an asset. You can hide the search box as part of the page customization. You can change the background color for the whole header section, and you can change the background color of the search box.

Below the header section, there can be one or more so-called widgets. Each widget contains a title, with an icon adjacent to the text. Under the title, you can have a list of entries, 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.

There are several kinds of widget:

- **Single-column widget**

  In this 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**

  In this 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**

  In this 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, i.e. there is an implicit HTML <td> element enclosing the HTML code you supply.

The Welcome page can contain up to 10 widgets. The widgets are displayed side by side in a single row.

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

Technical Implementation of the Welcome Page

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

Overview of Java Methods Used

The layout and contents of the Welcome page are implemented as Java code.

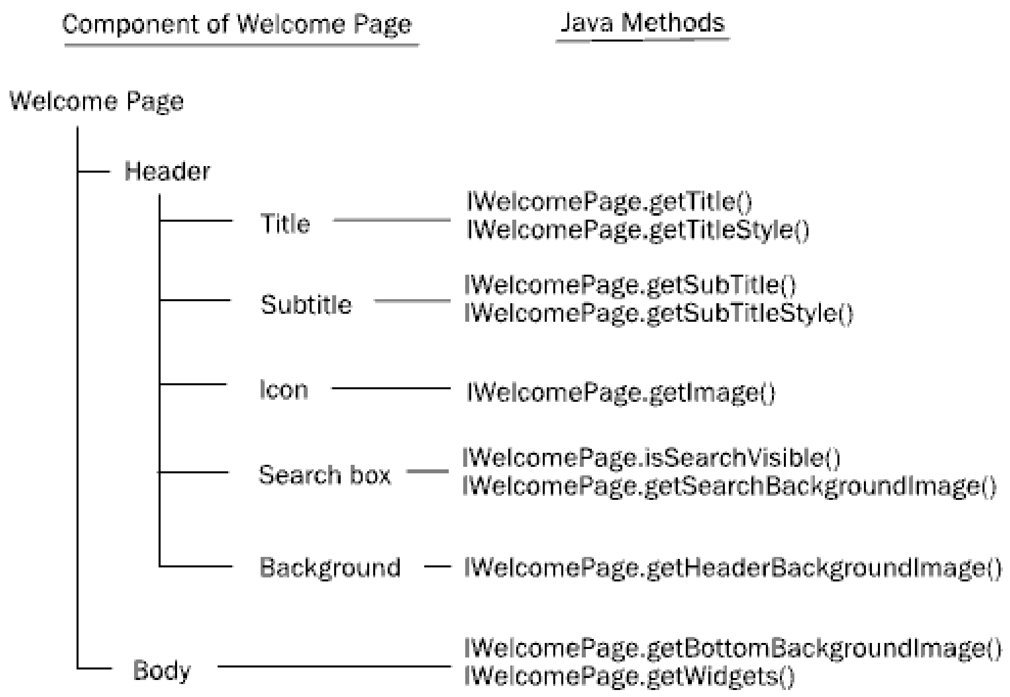
Each customizable part of the Welcome page requires a corresponding Java class. The Welcome screen can be defined as a combination of the following hierarchical structures:

- 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 shown in the following diagram.


Component of Welcome Page
Java Methods
Welcome Page
Header
Title
IWelcomePage.getTitle()
IWelcomePage.getTitleStyle()
Subtitle
IWelcomePage.getSubTitle()
IWelcomePage.getSubTitleStyle()
Icon
IWelcomePage.getImage()
Search box
IWelcomePage.isSearchVisible()
IWelcomePage.getSearchBackgroundImage()
Background
IWelcomePage.getHeaderBackgroundImage()
Body
IWelcomePage.getBottomBackgroundImage()
IWelcomePage.getWidgets()

The following table describes the purpose of these Java methods:

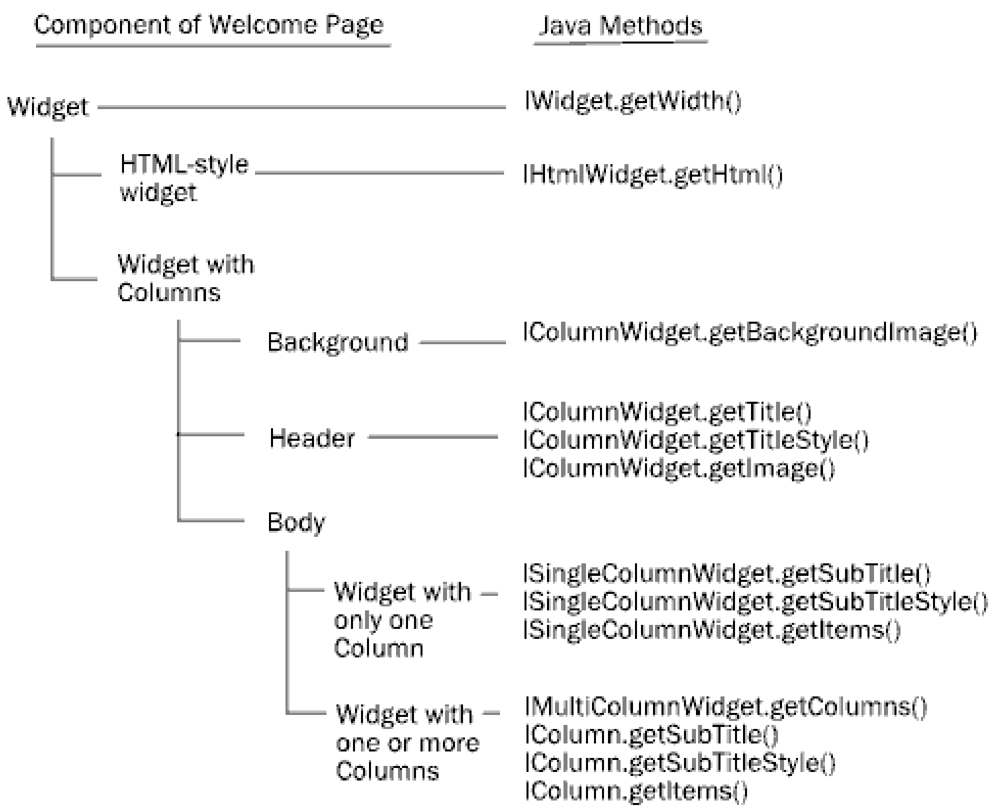
Name of Java interface	Java Method	Description
IWelcomePage	getTitle();	Defines the header text to be used.
IWelcomePage	getTitleStyle();	Defines the CSS style information for the header.
IWelcomePage	getSubTitle();	Defines the header subtitle text to be used.
IWelcomePage	getSubTitleStyle();	Defines 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.

Name of Java interface	Java Method	Description
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.
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/layout components are: a background image, a header text, the definition of a single-column table of items, the definition of a multi-column table of items.

The content and appearance of these components are determined by the Java methods shown in the following diagram.


Component of Welcome Page
Java Methods
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 following table describes the purpose of these Java methods:

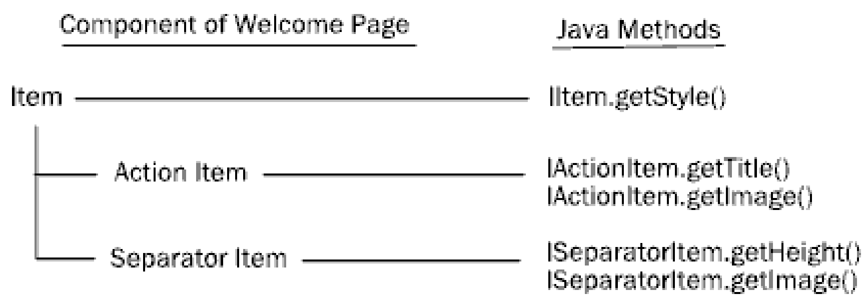
Name of Java interface	Java Method	Description
IWidget	getWidth();	Defines the screen width of the widget.
IHtmlWidget	getHtml();	Defines HTML code for an HTML-style widget.
IColumnWidget	getBackgroundImage();	Defines the background image to be used for a column of a widget.
IColumnWidget	getTitle();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

Name of Java interface	Java Method	Description
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, which 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 shown in the following diagram.



The following 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.

Methods Not Related to Screen Components

The following list shows the Java methods that are not related to a screen component, but which are required for the pluggable UI architecture of CentraSite Control.

Name of Java interface	Java Method	Description
(all interfaces)	setLocale();	This informs the widget or item about the CentraSite Control locale that is required to localize texts for the display. This method is called automatically before any other method that might depend on the locale.
(all interfaces)	setActionContext();	This informs the widget or item about the CentraSite Control context

Name of Java interface	Java Method	Description
		that is required for the processing to be done subsequently by the execute() method. This method is called automatically before any other method that might depend on the action context.
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 your customized Welcome page, you need to modify CentraSite Control's pluggable UI configuration in the Software AG Runtime environment.

Stop Software AG Runtime

Before you make any changes to the Software AG Runtime environment, stop the Software AG Runtime process.

Updating the plugin.xml Configuration File

The standard plugin.xml configuration file delivered with the CentraSite kit contains all of 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, and looks like this:

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

The second part defines the Java class that implements the Welcome page, and looks like this:

```
 <!-- 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 your customized Welcome page instead of the standard Welcome page, set the `class` attribute to your customized Java class that implements the interface IWelcomePage.

For general information about plugin.xml, see "Installing and Uninstalling Plug-Ins" on page 186.

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 might want to retain a copy of the original element and comment it out. This means that you can revert easily to the original Welcome page if required, by commenting out your customized element and uncommenting the original element.

Deploying the New Java Classes to the PluggableUI Environment

In addition to modifying the plugin.xml file, as described above, you need 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 your 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. If, for example, 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 via 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 need 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.

Start Software AG Runtime

After you have made the changes to the Software AG Runtime environment, restart the Software AG Runtime process. The changes you have made should now be visible when you view the Welcome Page.

Example of a Customized Welcome Page

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

Location of Demo Files

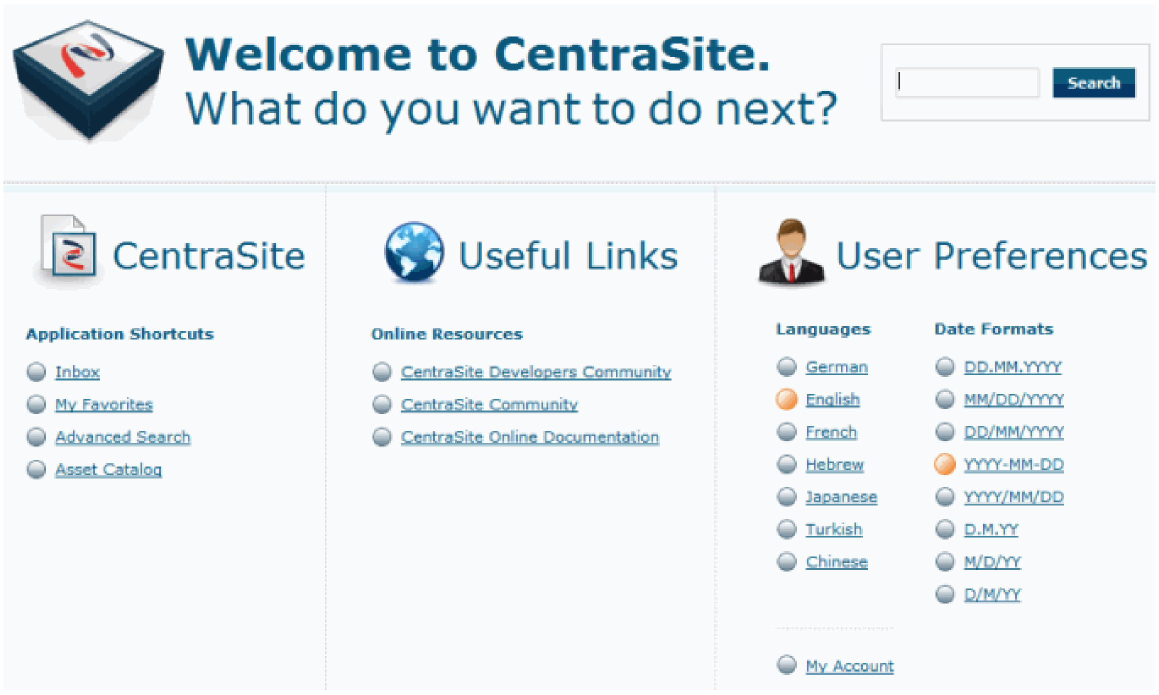
All of the required files for the demo are contained in the folder demos\WelcomePage under the CentraSite installation location. The following files are available at this location:

- 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 will be deployed to Software AG Runtime.

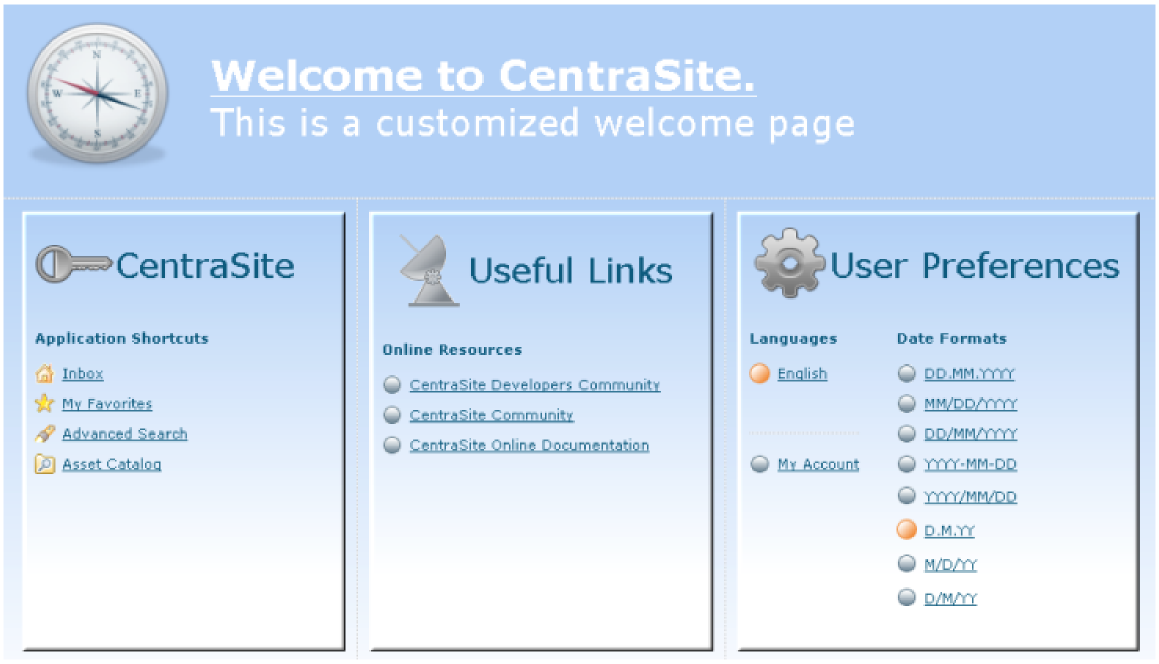
Differences Between the Standard Welcome Page and the Demo Welcome Page

This section shows the differences between the standard welcome page and the demo welcome page. Based on this you should be able to quickly evaluate the usefulness of this feature for your own business requirements.

The standard welcome page has the following appearance:



The demo welcome page used as an example in this section has the following appearance:



The main changes between the standard welcome page and the customized welcome page that Software AG supplies as a demo can be summarized as follows:

- 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 the 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();

Layout component	Source file	Java Method
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();
CSS style of subtitle text	CentraSiteWidget.java	getSubTitleStyle();
Header icon	CentraSiteWidget.java	getImage();

Layout component	Source file	Java Method
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();

Layout component	Source file	Java Method
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();
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();
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

Implementing the Demo as an Eclipse Java Project

If you wish 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. To create and use the Eclipse Java project, proceed as follows:

To create and use the Eclipse Java project

1. Start Eclipse.
2. Select **File > New > Project > Java Project**.

   This opens the wizard for creating a new Java project.
3. Select **Create project from existing source**.
4. 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.

Building the Deployment Files for Software AG Runtime

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

You can build the jar file by using Apache Ant with the build file build.xml provided in the demos\WelcomePage folder. 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 unzip the contents of the zip file directly into your Software AG Runtime location.

The build.properties File

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

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

Building the Deployment Files

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 will be copied to. In the extract shown above, Ant will search for all PNG graphics files ("*.png") in the resources folder in the build environment and add them to the zip file so that they can be unzipped into the images folder in the Software AG Runtime environment.

To build the deployment files, you can use either Eclipse or the command line.

In both methods, the Ant tasks defined in 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 build 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 (Method 1)

To build the deployment files (method 1)

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 (Method 2)

To build the deployment files from the command line

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

Deploying the Demo to Software AG Runtime

To deploy the demo Welcome page to Software AG Runtime, you need to copy the Java classes and icons of the demo Welcome page to the Software AG Runtimet environment, and update the Software AG Runtimeplugin.xml file. To do this, proceed as follows:

To deploy the demo Welcome page to Software AG Runtime

1. Stop Software AG Runtime.
2. Do one of the following alternatives:
   - Unzip the zip file created by the Ant build into <*RuntimeDir*> \workspace \webapps\PluggableUI\CentraSiteControl directory.

     This will copy 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.

     -- OR --
   - 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.
3. 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 "Updating the plugin.xml Configuration File" on page 154. 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.
4. Restart Software AG Runtime.

Displaying 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 will be visible when you start CentraSite Control.

Special Programming Techniques

This section summarizes some of the techniques you might find useful when creating your own customized welcome page. You can find code examples of the techniques in the demos\WelcomePage folder.

Technique	Code Example in demos \WelcomePage folder
Activate the `Advanced Search` page.	AdvancedSearchItem.java

Technique	Code Example in demos \WelcomePage folder
Activate the `Keyword Search` page.	KeywordSearchItem.java
Start the `My Account` dialog.	UserPreferencesItem.java
Start the `Add Asset` dialog.	CreateAssetItem.java
Start the `Import` dialog.	ImportWsdlFileItem.java
Activate `My CenstraSite` 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

Customizing Content Pages

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.

I18N for Layouts

Usage	Use this when the layout of a plug-in needs to be localized.
Attributes	■ `"point=com.softwareag.cis.plugin.i18n"` ■ `id` ■ `class` ■ `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 I18Message 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 I18Manager will attempt 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

```
<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>
```

Parameters for Plug-ins

Usage Use this to get parameters for a plug-in.

Attributes

- `point="com.softwareag.cis.plugin.parameter"`
- `id`
- `value`

Interface No interface to be implemented.

Processing Use the plug-in call ApplicationContext.getParameter() to obtain value.

Provided by PluggableUI

Example

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

ConnectionHandler - Logon and Logoff / Exit

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

Attributes

- `point="com.softwareag.cis.plugin.connectionHandler"`
- `id`
- `value`

Interface ConnectionHandler

- `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

- Logon:
  - Obtain credentials from the login screen
  - Call connect(Credentials) for each extension
  - If an exception occurs:
    - Show a popup with the exception
    - Disconnect each extension which is already connected
    - Restart
  - If all successful: start the workplace
- Logoff:
  - Call prepareDisconnect() for each extension
  - If an exception occurs:
    - Show a popup with the exception
    - Done
  - Disconnect each extension which is already connected by calling the disconnect() method

Provided by

PluggableUI

Example

```
<extension point="com.softwareag.cis.plugin.connectionHandler"
  id="login"
  class="com.centrasite.control.ext.CentraSiteConnectionHandler">
</extension>
```

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 will only be shown 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 in two ways:

- Select one or more rows (perspectives) and click **OK**.
- Double click 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 will be changed 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 via extensions. You can switch a perspective via the **Select Perspective** dialog.
- A fixed set of topics per perspective. The association between topics and the corresponding perspective is established via 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 will only be used if the perspective is used as the initial perspective. For more information, see "Setting the Preferred Plug-In and Order of Plug-Ins" on page 186.

Usage

Each plug-in may contribute a perspective to contain its own topics or the topics of other plug-ins

Attributes

- `point="com.softwareag.cis.plugin.perspective"`
- `id`
- `class`

Interface

- Perspective
  - `String getTitle()`

    (used in dynamically generated **Select Perspective** dialog)
  - `String getImageURL()`

    (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

- `point="com.softwareag.cis.plugin.topic"`

- `id`
- `perspective` (see "Perspectives" on page 169)
- `class`

Interface

Topic

- `String getImageURL()`
- `boolean isVisible()`

  (used when switching views)

Abstract base class

AbstractTopic

Processing

- 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

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

Command for Item

Usage

Add a command to a menu.

Attributes

- point="com.centrasite.control.itemCommand"
- 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

- When a list of commands for a menu item is retrieved (e.g. 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

- When a list of commands for a menu item is retrieved (e.g. 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.itemBulkCommand"
  id="test"
  class="com.centrasite.control.extpt.junit.
  DisplayRegObjKeyCommand">
</extension>
```

Add Property

Usage

Add a property to a registry object.

Attributes

- `point="com.centrasite.control.registryObjectProperty"`
- `id`
- `class`
- `(boolean)` visible by default

Interface

ExtensionPropertyAccessor

- `boolean appliesTo (String objectTypeQName, Connector con)`
- `String getDisplayName(Locale locale)`
- `String getDescription(Locale locale)`
- `String getInternalName()`
- `boolean getVisibleByDefault()`
- `String getValue(Item item) throws Exception`
- `void setValue(Item item, String value) throws Exception`

Abstract base class

AbstractPropertyAccessor (must be explicitly implemented)

Processing

- 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 might 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

- `point="com.centrasite.control.detailViewTab"`
- `id`
- `class`

Interface DetailViewTab

- `String getTitle()`
- `String getImageURL()`
- `String getLayout()`
- `void initAdapterFor (Item, DetailViewTabAdapter)`
- `protected String getAdapterClass();`
- `boolean appliesTo (Item)`

  (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)
- `void setDetailsTabContext (DetailTabContext)`
- `boolean isVisible(Item)`

Abstract base class AbstractDetailViewTab

Processing

- If the detail view for an Item is opened, it is checked for each known extension.
  - 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 will be 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

```
<extension point="com.centrasite.control.detailViewTab"
  id="lifecycle"
  class="com.centrasite.control.lifecycle.LifeCycleDetails">
</extension>
```

Add Source of Notification

Usage Add a source of a notification.

Attributes

- `point="com.centrasite.control.addRowToMyNotifications"`
- `id` (default: name of implementing class)
- `class`

Interface ReportExtensionItemsProvider

- `Collection getItems() throws Exception;`
- `void setConnector(Connector connector);`
- `boolean isContributedItem(Item item);`
- `String getChangedImageURL (Item item);`

Abstract base class AbstractReportExtensionItemsProvider

Processing

- The extension is initialized via the setConnector() method.
- Obtain all items to be added to the list of items with pending notification via the getItems() method.
- isContributedItem() can be used to check whether this extension has contributed the given item via getItems().
- getChangedImageURL() is used to control the icon representing the reason for the notification.

Provided by CentraSite Control

Example

```
<extension
  point="com.centrasite.control.addRowToMyNotifications"
  id="MyNotificationsApprovalItemsProvider"
  class="com.softwareag.centrasite.control.lms.ext.
```

```
                 MyNotificationsApprovalItemsProvider">
</extension>
```

Impact Analysis: NodeDecorator

Usage Change the visual representation of registry objects

Attributes

- `point="com.centrasite.control.assocNavigatorNodeDecorator"`
- `id` (default: name of implementing class)
- `class`

Interface NodeDecorator

- `String getImageURL(Item)`

Abstract base class (none)

Processing If the item is to be rendered in Impact Analysis, check all known extensions to determine whether they contribute to the item's visualization;

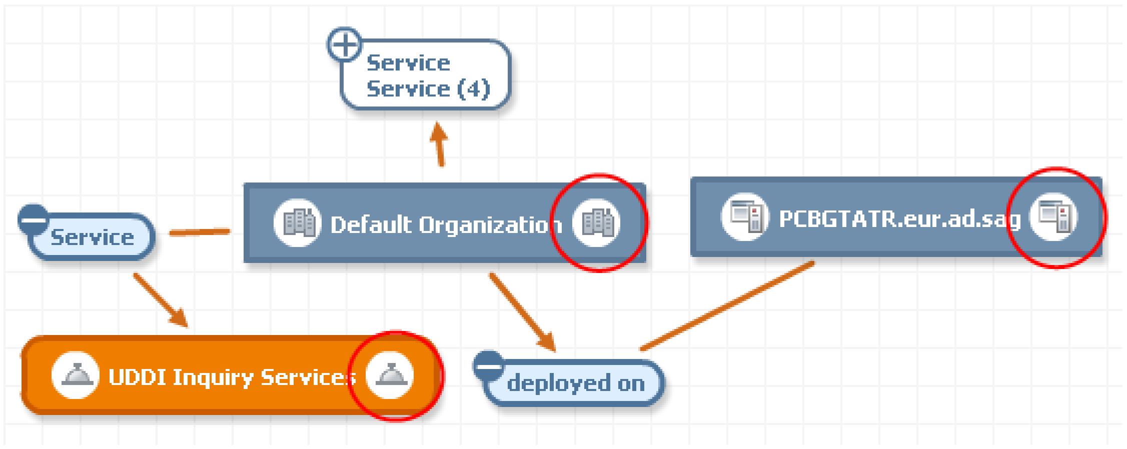
- If getImageURL(item) returns null: check for the next extension
- Otherwise: use the URL being returned for secondary icon within visualization of node in graphical impact analysis.

Provided by CentraSite Control

Example

```
<extension
  point="com.centrasite.control.assocNavigatorNodeDecorator"
  id="ExternalLinkNodeDecorator"
  class="com.centrasite.control.ext.ExternalLinkNodeDecorator">
</extension>
```

The following 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

- `point="com.centrasite.control.topicItems"`
- `id`
- `class`

Interface

TopicItems

- `boolean appliesTo (Topic)`
- `Collection getItems()`

Abstract base class

AbstractTopicItems

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 appliesTo() method returns `true`, the collection of Item objects returned by getItems() is appended to the set of root nodes for the corresponding topic.

Provided by

CentraSite Control

Example

```
<extension point="com.centrasite.control.topicItems"
  id="filesystem"
  topic="com.centrasite.control.administration"
  class="com.centrasite.control.ext.junit.
  FileSystemTopicItems">
</extension>
```

Note: Here, the `FileSystemTopicItems` extension is an extension of the base class AbstractTopicItems whose appliesTo() method returns `true` 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.

Attributes

- `point="com.centrasite.control.itemEditor"`
- `id` (default: name of implementing class)
- `class`

Interface

ItemEditor

- `public boolean appliesTo (Item item, Connector connector);`
- `public String getLayout();`
- `public String getTitle(Item item);`
- `public String getAdapterClass();`

  (must return a class implementing the `ItemEditorAdapter` interface)

Abstract base class

AbstractItemEditor

Processing

If appliesTo() returns true, the editor will be used when opening the detail for the item being passed:

- 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 getTitle() is used as the label for the activity.

Provided by

CentraSite Control

Example

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

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

- `point="com.centrasite.control.searchPredicate"`
- `id`
- `class`

Interface

PredicateEditor

- `Predicate getPredicate()`

  (Get predicate to be added by this editor)
- `String getLayout()`

  (Get URL of layout to be rendered)
- `String getAdapterClass()`

  (Get name of adapter class to be used for rendering, must be a subclass of AbstractPredicateAdapter)
- `String getPredicateClass()`

  (Get name of predicate class to be used for rendering, must be a subclass of AbstractPredicate)

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 will display 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 predicateEement, Connection connection)`
- used to initialize the search dialog with readonly predicates / conditions which can neither modified or removed:
- `void setReadOnly(boolean readOnly);`
- `boolean isReadOnly();`

Abstract base class

AbstractPredicateEditor

Processing

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

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.

Making the Download Menu Entry Visible/Invisible for Guest Users

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/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 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`. If you want to change this, you can do so by extending the CentraSite Control functionality via 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	■ `com.centrasite.control.downloadDocumentCommand`

- `id` (default: name of implementing class)
- `class`

Interface	See the sample code.
Abstract base class	AbstractExtensionCommand
Provided by	CentraSite Control
Example	See the sample code.

To change the text displayed for the context menu, your 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. If you want to change this, you can do so by extending the CentraSite Control functionality via the extension point downloadDocumentCommand. The definition of the extension point is given above.

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. When trying to attach a supporting document with an asset, CentraSite Control displays the available documents underneath their respective organization directory by default. If you want to change this (that is, simply display the documents by the side of its organization directory), you can do so by extending the CentraSite Control functionality via 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 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 back 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 via the **Attach Document** dialog.
Attributes	■ com.centrasite.control.attachDocumentCommand ■ id (default: name of implementing class) ■ class
Interface	See the sample code.
Abstract base class	AbstractExtensionCommand
Processing	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. ■ Create an instance of the class ■ Execute
Provided by	CentraSite Control
Example	`<extension point="com.centrasite.control.attachDocumentCommand" id="AttachDocumentCustomCommand" class="com.centrasite.control.extpt.AttachDocumentCustomCommand" />`

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 demo 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 inside 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 the *CentraSite Administrator's Guide*.

Step-by-Step Guide

A step-by-step guide of how to create customized plug-ins for the CentraSite Control content pages is provided in "Step-by-Step Guide" on page 192.

Setting the 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 would 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 will be 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. The *CentraSite Administrator's Guide* describes the location of this directory.

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 described in "Directory Structure" on page 186.

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

- Check for availability of other plug-ins being a prerequisite.
- 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
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 relies heavily on dedicated class loaders. Whereas the code for a normal web application is only loaded via the basic `WebappClassLoader` provided by the servlet container (e.g. 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
```

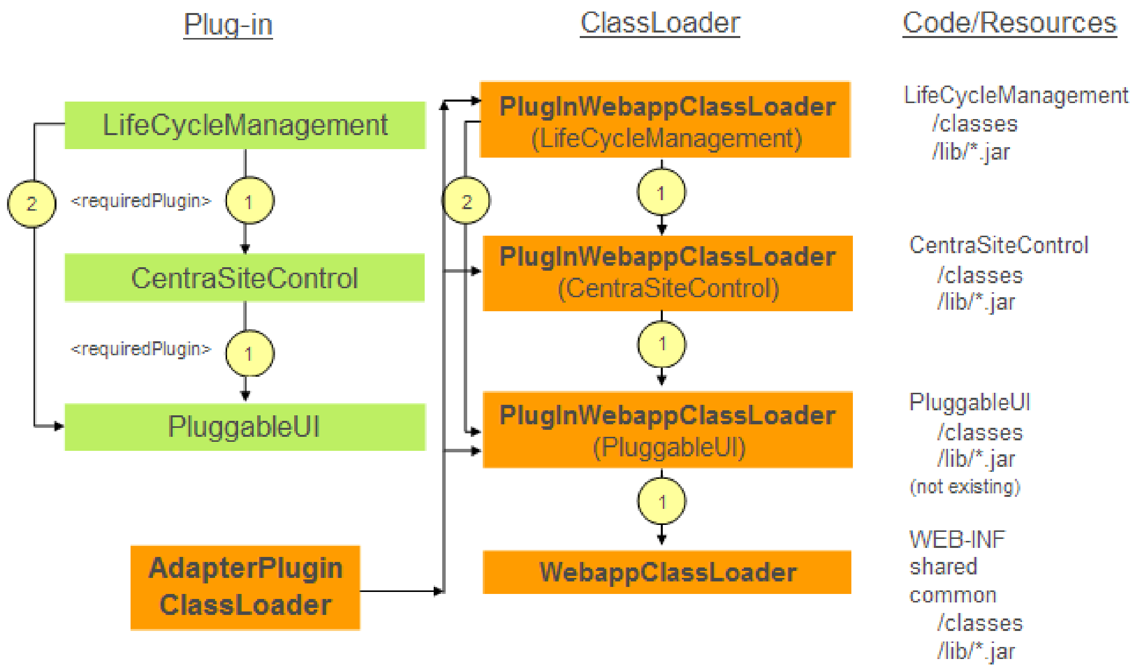
In addition, locations holding common or shared class or jar files are searched by the `WebappClassLoader` when attempting to resolve references to required classes.

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

The following picture illustrates the scenario of the LifeCycleManagement plug-in (representing any other 3rd party plug-in) that depends on the CentraSiteControl plug-in and the PluggableUI base infrastructure.


Plug-in
ClassLoader
Code/Resources
LifeCycleManagement
<requiredPlugin>
1
2
CentraSiteControl
<requiredPlugin>
1
PluggableUI
AdapterPlugin ClassLoader
PlugInWebappClassLoader (LifeCycleManagement)
1
2
PlugInWebappClassLoader (CentraSiteControl)
1
PlugInWebappClassLoader (PluggableUI)
1
WebappClassLoader
LifeCycleManagement /classes /lib/*.jar
CentraSiteControl /classes /lib/*.jar
PluggableUI /classes /lib/*.jar (not existing)
WEB-INF shared common /classes /lib/*.jar

The `AdapterPlugInClassLoader` is used by ApplicationDesigner when resolving references to adapter classes found in layout definitions. The `AdapterPlugInClassLoader` will never load classes itself. Instead, it will ask all known `PlugInWebappClassLoaders` in an unspecified order whether they can load a required class.

Caution: You must avoid having the same adapter classes in more than one plug-in! Otherwise, various classloading related issues (ClassCastException, ClassNotFoundException, …) will result. 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 / static variables, this is no longer true. Multiple active user sessions may be processed in parallel.

Warning About Using Global Variables

Avoid usage of global variables, which might lead to the following issues:

- Synchronization is required otherwise non-reproducible race conditions will result.
- 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 will be used, thus causing a security hole.
  - If the connection is no longer open, a corresponding exception will be thrown (`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 e.g. 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

Javadoc Documentation of the APIs

The *CentraSite Java API Reference* provides details of the classes and methods described here. This document is available at Software AG Documentation Website.

You should only use the packages and classes that are explicitly mentioned in the documentation.

There are three sets of Javadoc documentation:

- PluggableUI (base architecture): *com.softwareag.cis.plugin*
- CentraSite Control User Interface: *com.centrasite.control...*
- CentraSite Control Backend: *com.centrasite.control...*

Step-by-Step Guide

Eclipse Prerequisites

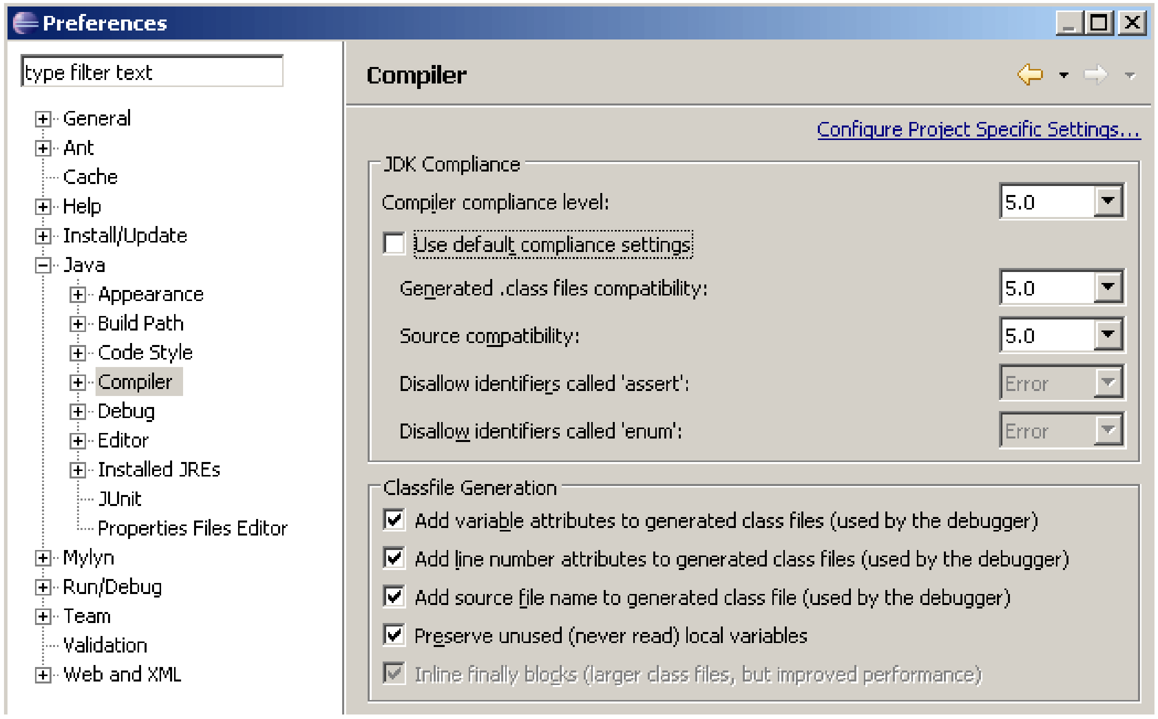
The descriptions in this topic are based on a sample plug-in named *DemoPlugIn01* . We will show how to use Eclipse and standard CentraSite features in order to add the plug-in to CentraSite.

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:



JDK Compliance

Compiler compliance level: 5.0

☐ Use default compliance settings

Generated .class files compatibility: 5.0

Source compatibility: 5.0

Disallow identifiers called 'assert': Error

Disallow identifiers called 'enum': Error

Classfile Generation

☑ Add variable attributes to generated class files (used by the debugger)

☑ Add line number attributes to generated class files (used by the debugger)

☑ Add source file name to generated class file (used by the debugger)

☑ Preserve unused (never read) local variables

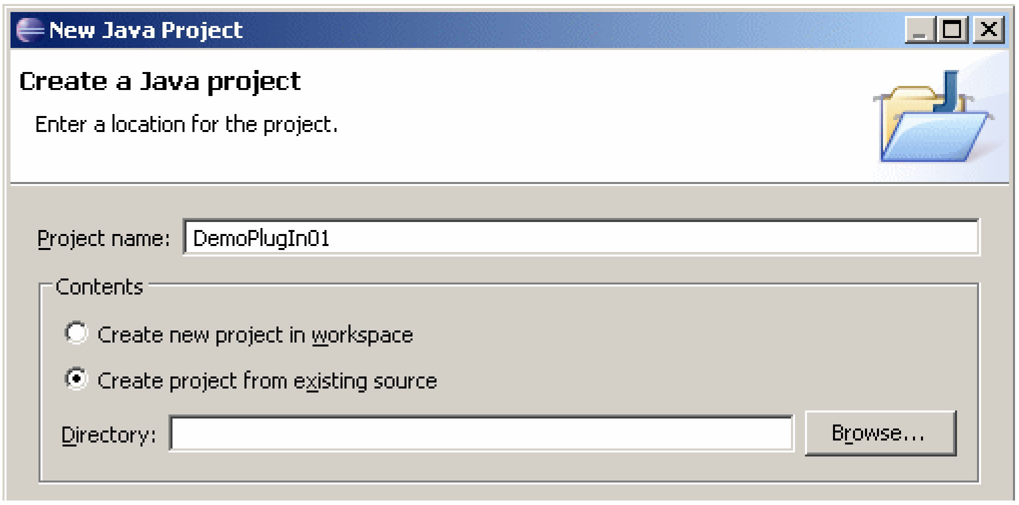
☑ Inline finally blocks (larger class files, but improved performance)

Click **Apply** to activate the settings. Eclipse will ask you to confirm the change, indicating that an internal rebuild is required. Reply **Yes**. The rebuild takes only a few seconds.

Setting up the Plug-in Project

Follow the steps below to set up the Java project for DemoPlugIn01:

1. Create a new Java project in Eclipse using **File > New > Project > Java Project**.
2. Specify `DemoPlugIn01` as the project name and check radio button labeled **Create project from existing source**.

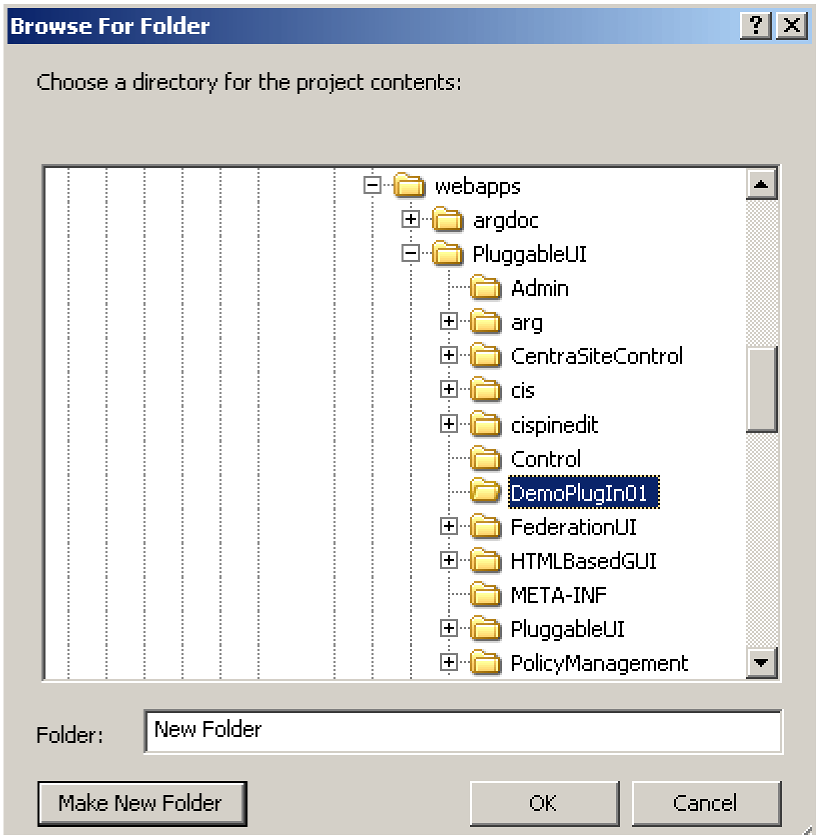


Project name: DemoPlugIn01

( ) Create new project in workspace
(●) Create project from existing source

Directory:

3. Click the **Browse** button that is located right to the input field labeled **Directory**. The **Browse For Folder** dialog is displayed.
4. Within 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>* .
5. Click **Make New Folder**. This causes an entry New Folder to be created under PluggableUI. Select the entry New Folder, then from its context menu choose **Rename**, then enter the name DemoPlugIn01.



6. Click **OK**.
7. In the **New Java Project** dialog that becomes visible again, click **Finish**.

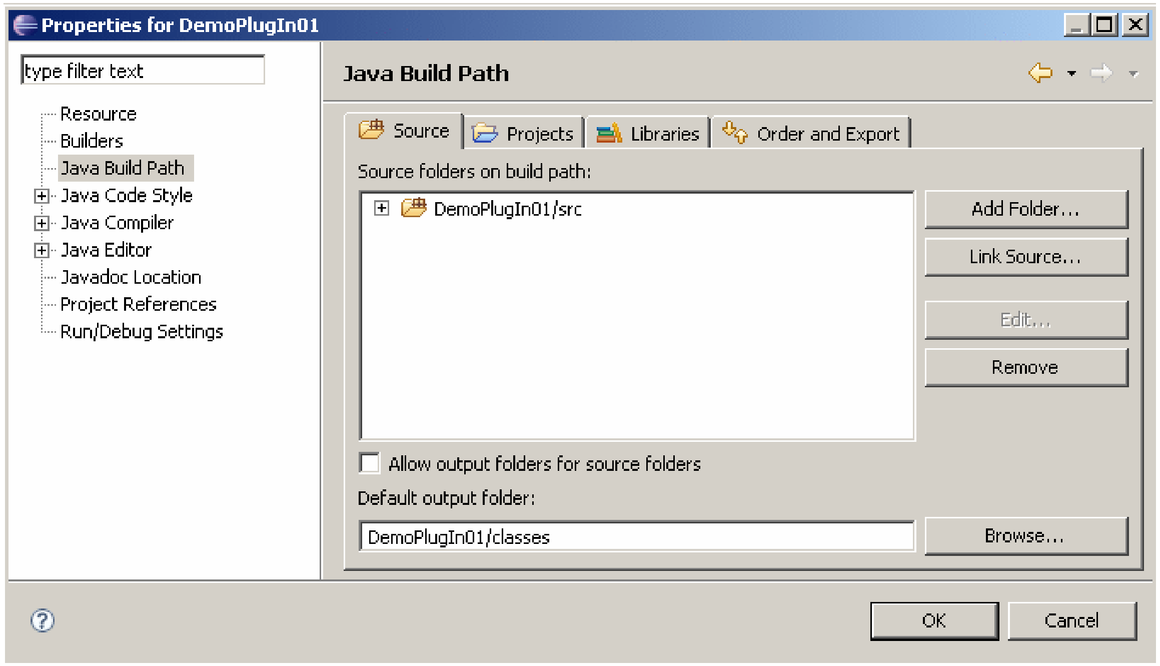
A new Java project called DemoPlugIn01 has been created due to the previous actions. This project is now visible in the **Package Explorer** view in Eclipse.

This project needs to be adapted.

1. Create the following four subfolders of DemoPlugIn01:
   - accesspath
   - classes
   - images
   - xml

   To create each of these subfolders, choose **New > Folder** from the context menu of DemoPlugIn01 in the package explorer, then type in the name in the **New Folder** dialog.
2. Create a source folder called src. You can create the source folder by choosing **New > Source Folder** from the context menu of DemoPlugIn01.
3. In the context menu of DemoPlugIn01, choose **Properties**.
4. Select **Java Build Path** from the tree on the left.

5. Select the **Source** tab and enter the value `DemoPlugIn01/classes` in the field **Default output folder**.



Source folders on build path:
DemoPlugIn01/src

Allow output folders for source folders

Default output folder:
DemoPlugIn01/classes

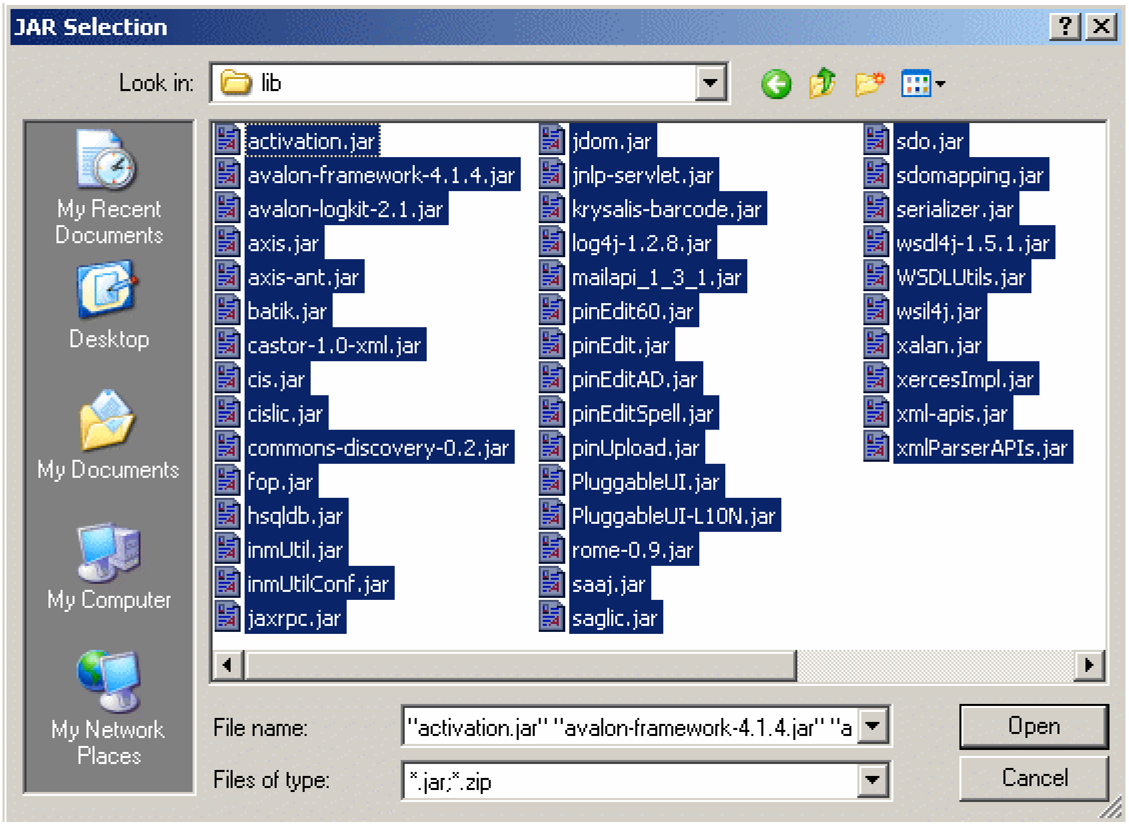
6. Switch to the **Libraries** tab.

   An entry for the JRE library should be visible. If you do not see this entry, click **Add Library** and select the JRE system library from the displayed list, then click **Finish**.

7. Click **Add External JARs**

   In the resulting **JAR Selection** dialog, navigate to <*PluggableUIFolder*> /WEB-INF/lib and open this folder.

8. Select all files, using for example the key combination Control-A, and click **Open**.

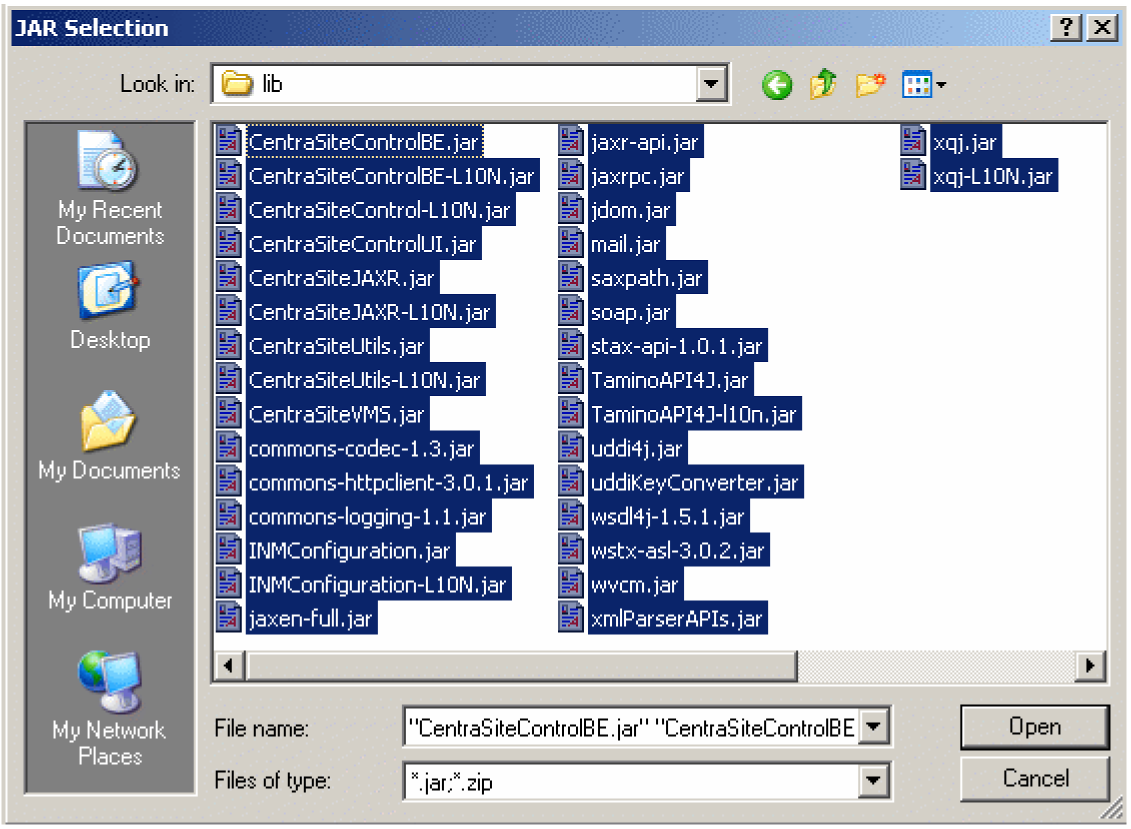


Look in: lib

File name: "activation.jar" "avalon-framework-4.1.4.jar" "a

Files of type: *.jar;*.zip

9. Click **Add External JARs** again.
10. In the **JAR Selection** dialog, navigate to <*PluggableUIFolder*> /CentraSiteControl/lib.
11. Again, select all files and click **Open**.



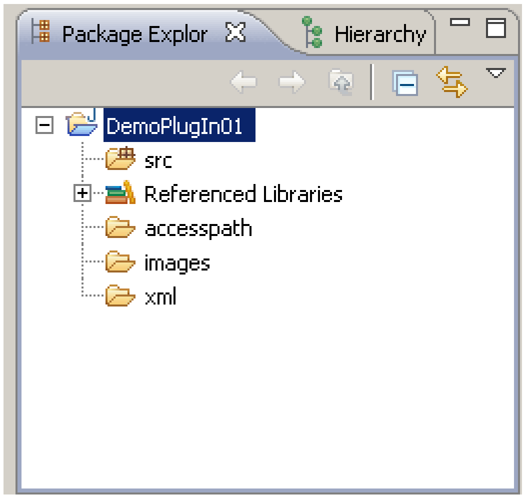
CentraSiteControlBE.jar
CentraSiteControlBE-L10N.jar
CentraSiteControl-L10N.jar
CentraSiteControlUI.jar
CentraSiteJAXR.jar
CentraSiteJAXR-L10N.jar
CentraSiteUtils.jar
CentraSiteUtils-L10N.jar
CentraSiteVMS.jar
commons-codec-1.3.jar
commons-httpclient-3.0.1.jar
commons-logging-1.1.jar
INMConfiguration.jar
INMConfiguration-L10N.jar
jaxen-full.jar
jaxr-api.jar
jaxrpc.jar
jdom.jar
mail.jar
saxpath.jar
soap.jar
stax-api-1.0.1.jar
TaminoAPI4J.jar
TaminoAPI4J-l10n.jar
uddi4j.jar
uddiKeyConverter.jar
wsdl4j-1.5.1.jar
wstx-asl-3.0.2.jar
wvcm.jar
xmlParserAPIs.jar
xqj.jar
xqj-L10N.jar

File name: "CentraSiteControlBE.jar" "CentraSiteControlBE

Files of type: *.jar;*.zip

12. Click the **OK** button of the **Properties for DemoPlugIn01** dialog.

Your project should now look like this:

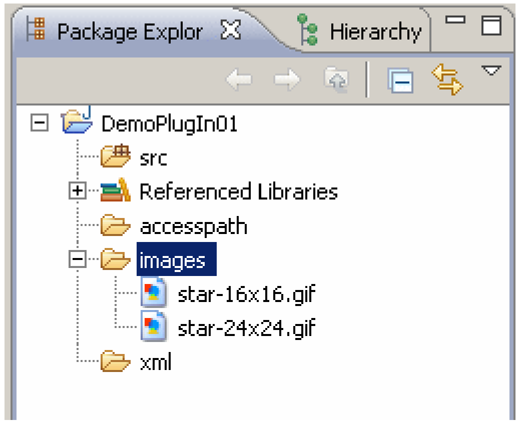


Perhaps you have noticed that the classes subfolder of the project DemoPlugIn01 has disappeared from the display. This is normal because the Java Development Tools (JDT) of Eclipse suppress output folders from displaying by default (but they still exist on your hard disk).

Furthermore, the old output folder bin that has been created by the JDT when creating the Java project is not of any use for us, so you can delete it.

Later on we will need some icons for our plug-in. For now, let's just copy and rename some already existing icons from the CentraSite Control plug-in and use them instead:

1. Using the Windows Explorer, navigate to <*PluggableUIFolder*> /CentraSiteControl/ images.
2. Copy the files myFavorites.gif and myFavorites24x24.gif to the images subfolder of our Java project DemoPlugIn01.
3. 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.
4. In Eclipse, refresh the display of the package explorer. The names of the two images should now be visible.



Plugging into CentraSite Control

We have now created a Java project inside the PluggableUI web application. However, there is one missing piece that tells CentraSite Control that this folder contains a plug-in: the plug-in configuration file. Amongst other things, the plug-in configuration file contains the information about where a plug-in plugs into in CentraSite Control.

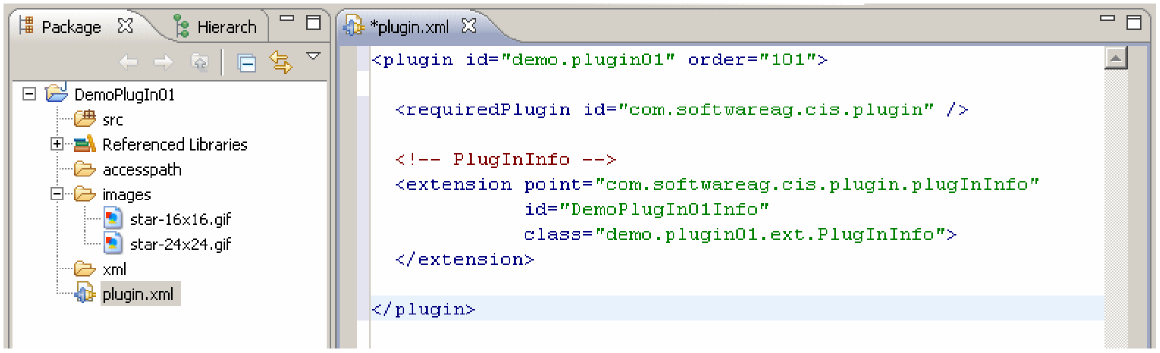
The idea of using plug-ins to extend an application's functionality is quite simple and meanwhile well established by the Eclipse platform. The CentraSite Control software provides so-called extension points. These are positions in the program logic of the CentraSite Control program where functionality 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.

Let's convert our arbitrary Java project to a CentraSite Control plug-in folder by providing a plug-in configuration file. To do so, follow the steps below:

1. In the context menu of DemoPlugIn01 in the package explorer, choose **New > File**.

2. Type plugin.xml as the file name and click **Finish**.
3. Enter 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>
```



```
<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>
```

4. Save the file using <Ctrl>+S.

First of all a plug-in must have an identifier (here `demo.plugin01`) which has to be unique among all plug-ins. We recommend you 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 need to declare our plug-in as being dependent on the plug-in com.softwareag.cis.plugin because we use an extension point provided by this plug-in. This dependency is indicated through the `requiredPlugin` XML element.

For a list of all supported extension points, see "Extension Points" on page 167.

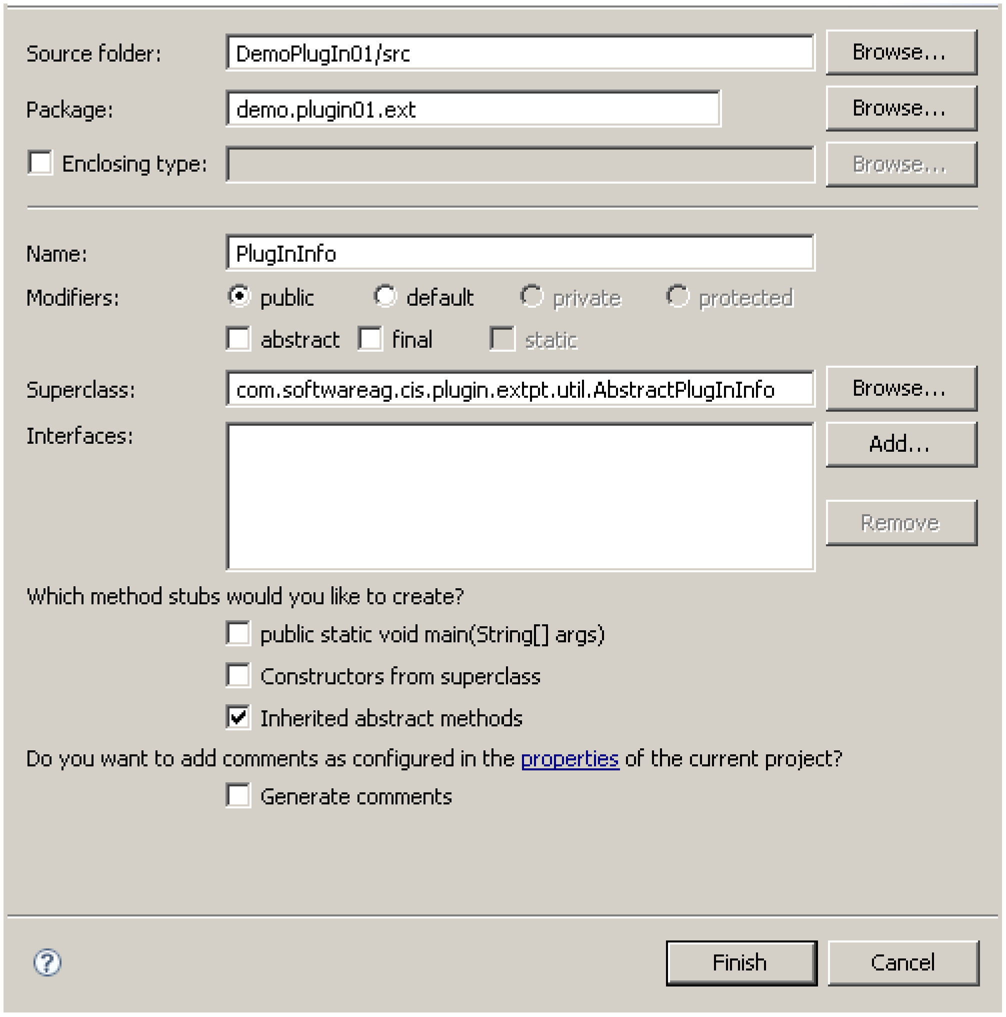
The `extension` XML element in our file DemoPlugIn01/plugin.xml denotes that our plug-in extends the user interface at a point where information about a plug-in can be contributed. The string that looks like a Java package name is the name of the extension point (com.softwareag.cis.plugin.plugInInfo).

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 will be demo.plugin01. We choose ext as the subpackage name for the implementing class to denote that code that extends CentraSite Control resides here.

Now we have to implement the extension, i.e. we have to provide a Java class called demo.plugin01.ext.PlugInInfo which implements a specific interface required by the extension point.

1. In the context menu of DemoPlugIn01/src in the package explorer, choose **New > Class**.
2. 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 may uses the **Browse** button to save some typing).
3. Make sure that the check box labeled with **Inherited abstract methods** is checked and click **Finish**.


Source folder: DemoPlugIn01/src
Package: demo.plugin01.ext
☐ Enclosing type:
Name: PlugInInfo
Modifiers: [✓ public] default private protected
☐ abstract ☐ final ☐ static
Superclass: com.softwareag.cis.plugin.extpt.util.AbstractPlugInInfo
Interfaces:
Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments as configured in the properties of the current project?
☐ Generate comments

Eclipse now opens the file PlugInInfo.java in the Java editor.

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";
   }
}
```

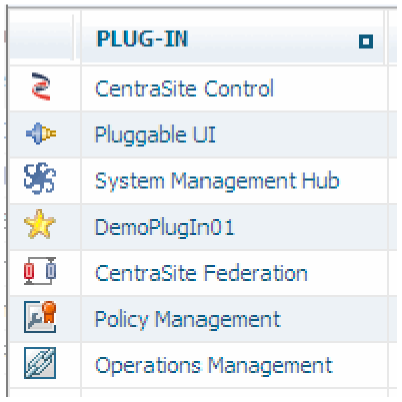
Save the modified file and make sure that no compile errors occur.

When you save the file, the Java file is automatically compiled into the folder classes of the project DemoPlugIn01. (Remember: the classes subfolder of our project is suppressed from displaying). The resulting class file is now accessible for the pluggable user interface of CentraSite Control.

Finally, let's check if CentraSite Control is aware of our minimalist plug-in:

1. Restart the Windows service "Software AG Runtime".
2. Start the CentraSite Control application from the Windows **Start > All Programs > Software AG** menu, and log in using your usual ID and password.
3. Click the **About** button at the top of the page. In the subsequent dialog, click **Plug-Ins**.

If everything works fine, you should see a dialog box whose contents look quite similar to the following screenshot. In particular, the line that represents our sample plug-in DemoPlugIn01 should be visible.



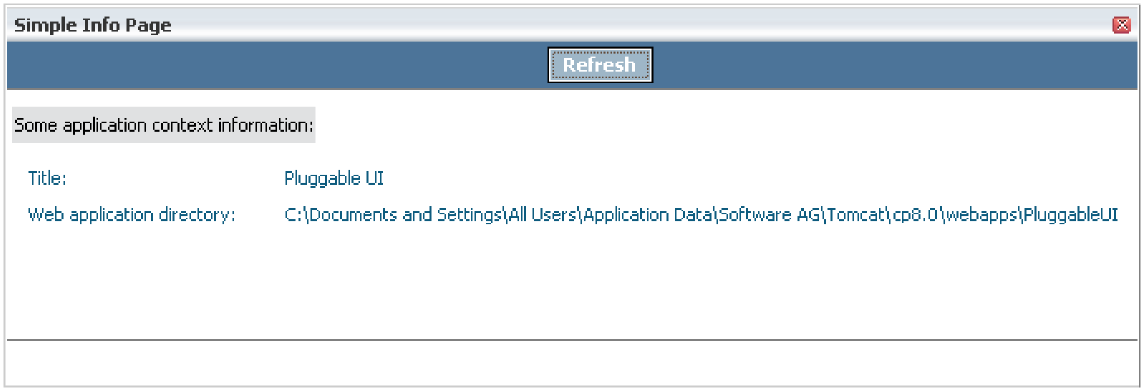
	PLUG-IN
	CentraSite Control
	Pluggable UI
	System Management Hub
	DemoPlugIn01
	CentraSite Federation
	Policy Management
	Operations Management

Bring Your Own Layouts to the Screen

You can extend CentraSite Control by embedding your own layout pages. In this step you will learn how to create a layout page with by using the Application Designer IDE. Furthermore you will learn a very simple way to bring your own layout onto the screen.

Before we start, let's preview what the result of this step will be. We will create a simple layout page that presents some information to the user. This page can be requested by the user by clicking on an icon in the tool bar of CentraSite Control. Feel free to extend the layout page and enhance it with more information after you have worked through this step. In subsequent steps we will use this page many times when we extend CentraSite Control at more and more extension points. So please make sure that this page and the code behind it is running properly.

Here is a screenshot of the final result of this step:



Some application context information:

Title: Pluggable UI

Web application directory: C:\Documents and Settings\All Users\Application Data\Software AG\Tomcat\cp8.0\webapps\PluggableUI

In order to create and design a layout page we need to have the right tool. The CentraSite distribution kit contains an IDE (integrated development environment) that you can use for this purpose. There is currently no shortcut created by the installation to start this IDE. Therefore we have to create one manually. The IDE is a web application whose clients run inside 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`.

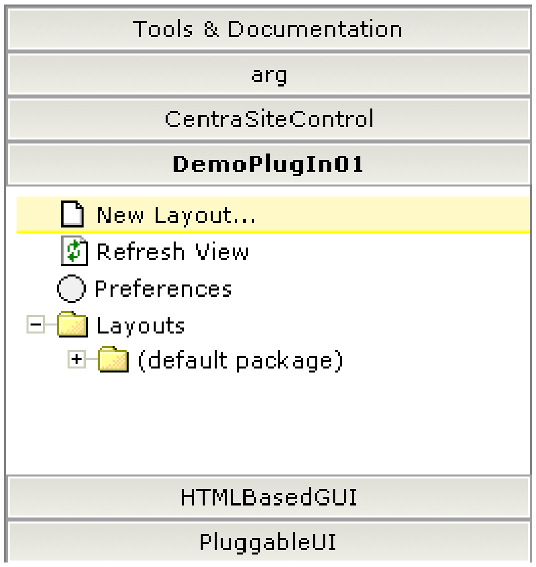
The IDE delivered in the distribution kit needs to be activated before you can use it. For information on how to do this, see "Activating the IDE" on page 185.

The documentation for the IDE is available at http://documentation.softwareag.com.

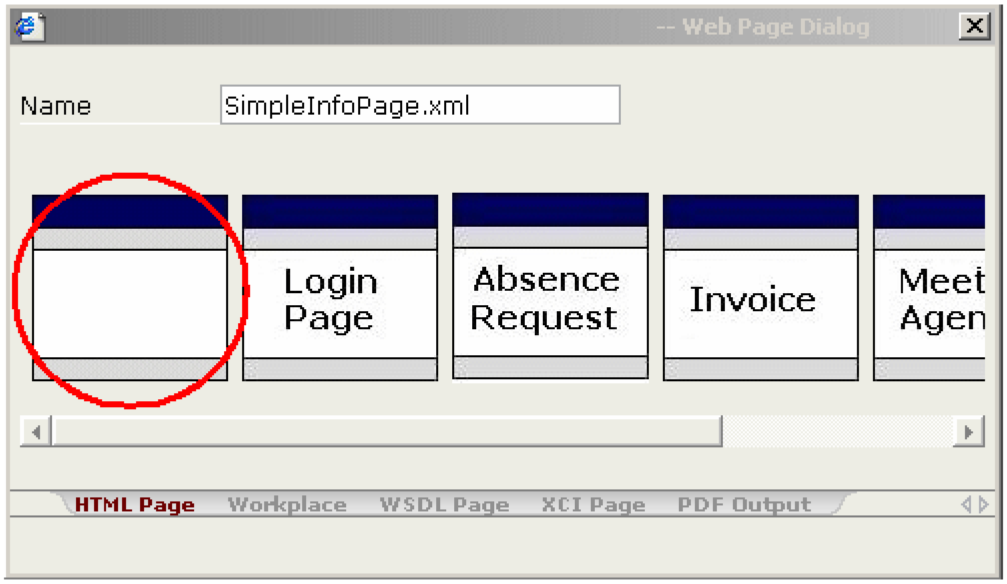
We start with the creation and design of a layout page. So please start the IDE now (leave your Eclipse instance running).

After the IDE has started, perform the following steps:

1. In the button list on the left, click the button labeled with the name of our plug-in **DemoPlugIn01**.

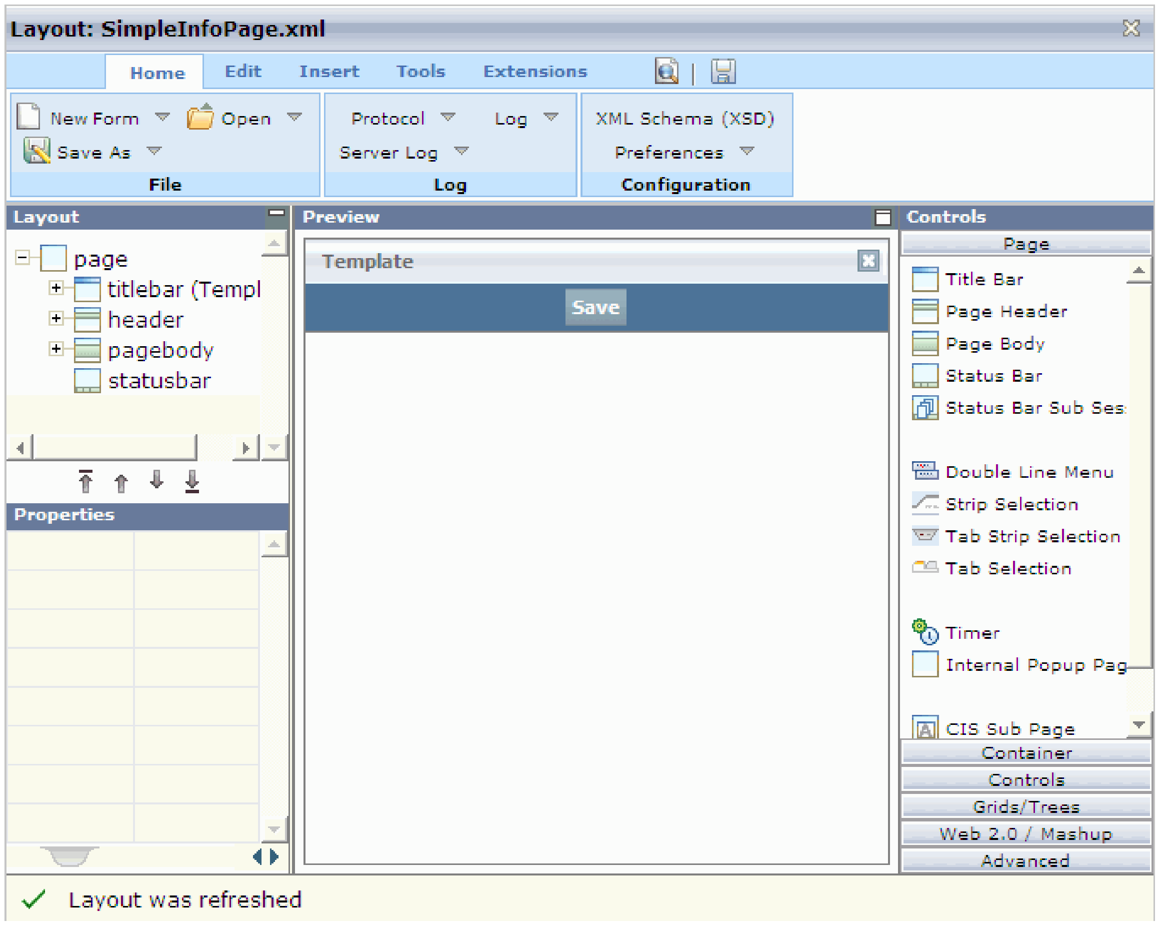


2. Click **New Layout** on the left side directly below the **DemoPlugIn01** button.
3. In the resulting dialog window enter `SimpleInfoPage.xml` in the input field labeled **Name**.
4. Click the leftmost image below the input field (see screenshot) to create an HTML page.



Name SimpleInfoPage.xml

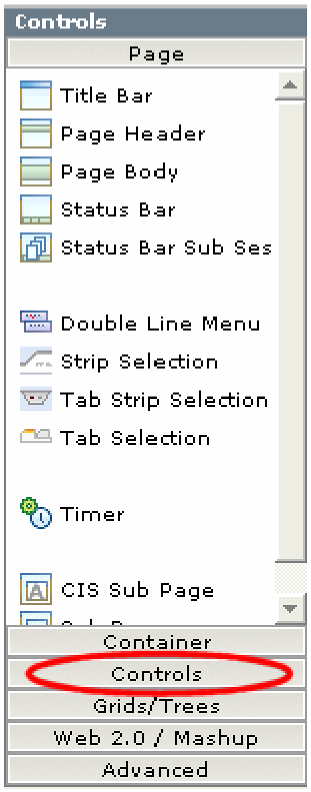
The IDE presents a standard HTML page in the preview area of the layout painter (in the center of the right side). To get an idea about how our newly created layout looks initially, we should request a preview of it from the layout painter. To do so, select the **Preview** icon from the toolbar of the layout painter (located beside the diskette symbol). The current look of layout SimpleInfoPage.xml is presented in the preview area.



Layout was refreshed

Starting from this layout, we will follow the steps below to create a layout that looks like the one that is shown at the beginning of this step:

1. Click on the title bar of our layout (the bar above the **Save** button where the word **Template** is visible).
2. 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`.
3. Click the **Save** button of our layout (the content of the **Properties** view changes and the current properties for the selected button become visible). Change the `name` property from `Save` to `Refresh` and set the `method` property to `onRefresh` by just typing it in.
4. Save the layout by clicking on the diskette symbol in the tool bar of the layout painter. The preview of our layout changes and now reflects the properties we changed.
5. Click the **Controls** button of the button list in the **Controls** view (located right to the preview area).



6. Add three **Independent Row** controls to the body of our page:

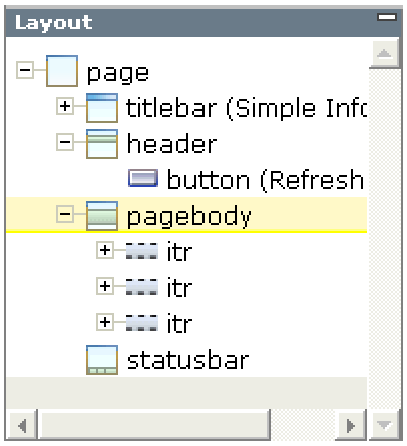
   a. Click **Independent Row** and hold the left mouse button down.

   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.

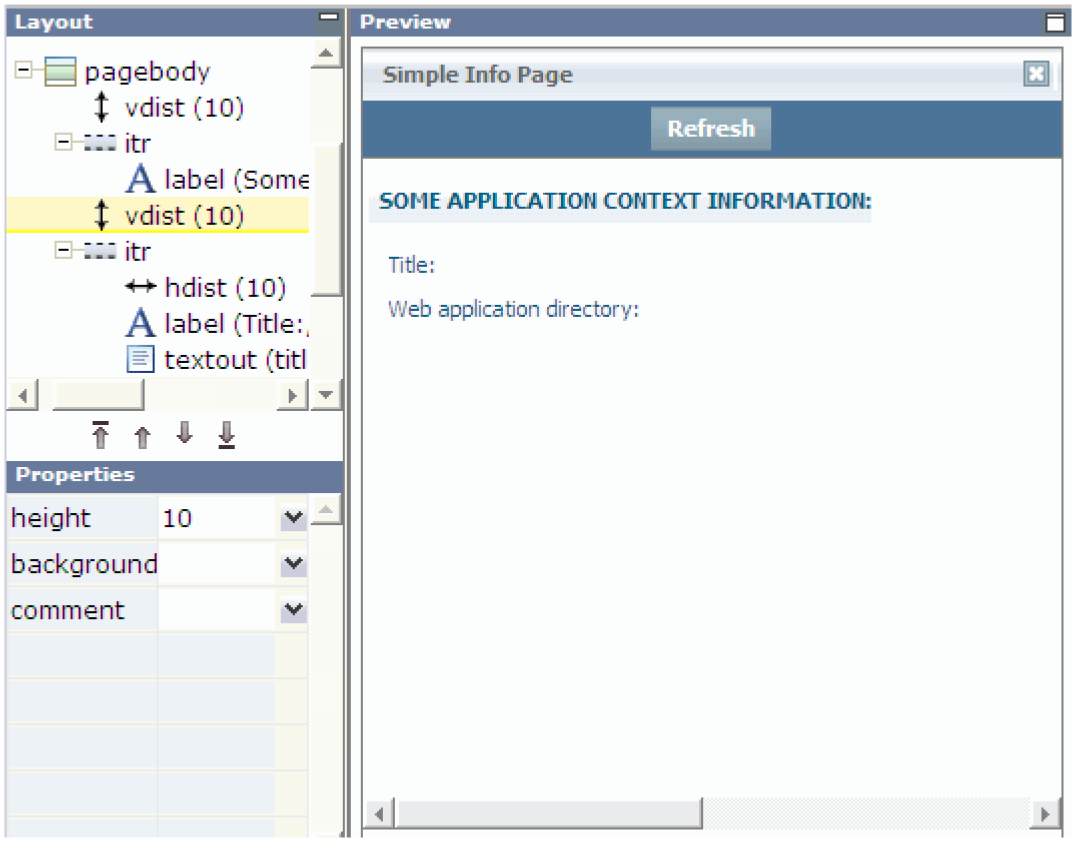
   d. Perform the same action to add a second and a third **Independent Row** control to the page body and click **Add as last Subnode** from the popup menu that appears after releasing the mouse button.

7. Please notice that the `pagebody` node of the **Layout** view (located above the **Properties** view) now contains three `itr` subnodes, representing the three **Independent Row** controls.



8. From the **Controls** view, drag and drop a **Label** control onto the first `itr` subnode of the `pagebody` node in the **Layout** view.
9. In the **Properties** view (which now presents the properties for the **Label** control we just added to the layout) set the `name` property to `Some application context information:`.
10. 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.
11. Drag and drop a **Horizontal Distance** control onto the second `itr` subnode of the `pagebody` node in the **Layout** view.
12. In the **Properties** view, set the `width` property for the **Horizontal Distance** control to `10` by just typing it in.
13. Drag and drop a **Label** control onto the second `itr` subnode of the `pagebody` node in the **Layout** view. From the popup menu that appears after you release the mouse button, click **Add as last Subnode**.
14. Set the `name` property of the newly added label to `Title:` and the `width` property to `200`.
15. Add a **Dynamic Text** control as the last subnode to the second `itr` subnode of `pagebody`. Set the `valueprop` property to `title` and the `width` property to `500`.
16. Execute the last five steps again for the third `itr` subnode 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` subnode of `pagebody`.
17. Surround the first `itr` subnode with two vertical distances by dragging and dropping two **Vertical Distance** controls onto the first `itr` subnode 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`.

18. Save the layout by clicking again on the diskette symbol in the tool bar of the layout painter.



height	10
background	
comment	

SOME APPLICATION CONTEXT INFORMATION:

Title:

Web application directory:

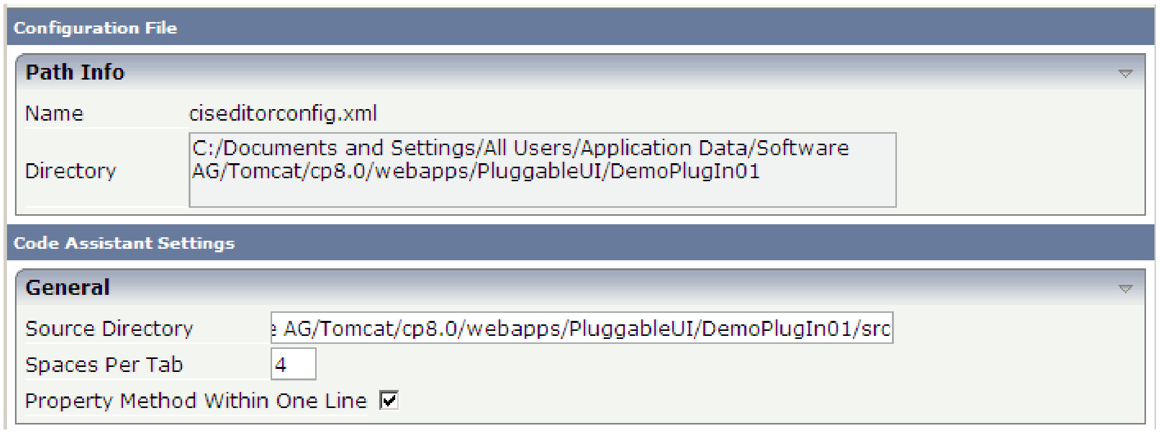
Now our layout looks like the one that is shown at the beginning of this step. But we are not finished yet. Each layout needs to have some code behind it (the so-called page adapter) which we did not provide yet. Among other things within a page adapter we can specify how to react on events that occur due to user interactions (the push of a button for example) or fill the controls with application specific values etc.

The code behind our layout at this stage is provided by a dummy adapter that comes with the IDE. But we need to provide our own, of course, so that the adapter knows what to do when a user presses the **Refresh** button, for example. So our next task is to create an adapter for our layout. Fortunately the IDE is equipped with tools that make life easy.

Follow the steps below to create an adapter for SimpleInfoPage:

1. If not already active, switch to the **Home** tab of the layout painter.
2. Select **Preferences** and type in the absolute path for the Java source directory of our CentraSite Control plug-in DemoPlugIn01.

> **Tip:** Instead of typing in the complete path, you can copy/paste the content of the field labeled **Directory** into the input field for the source directory. Then append `/src` to the copied content (compare with following screenshot).



Configuration File

Path Info

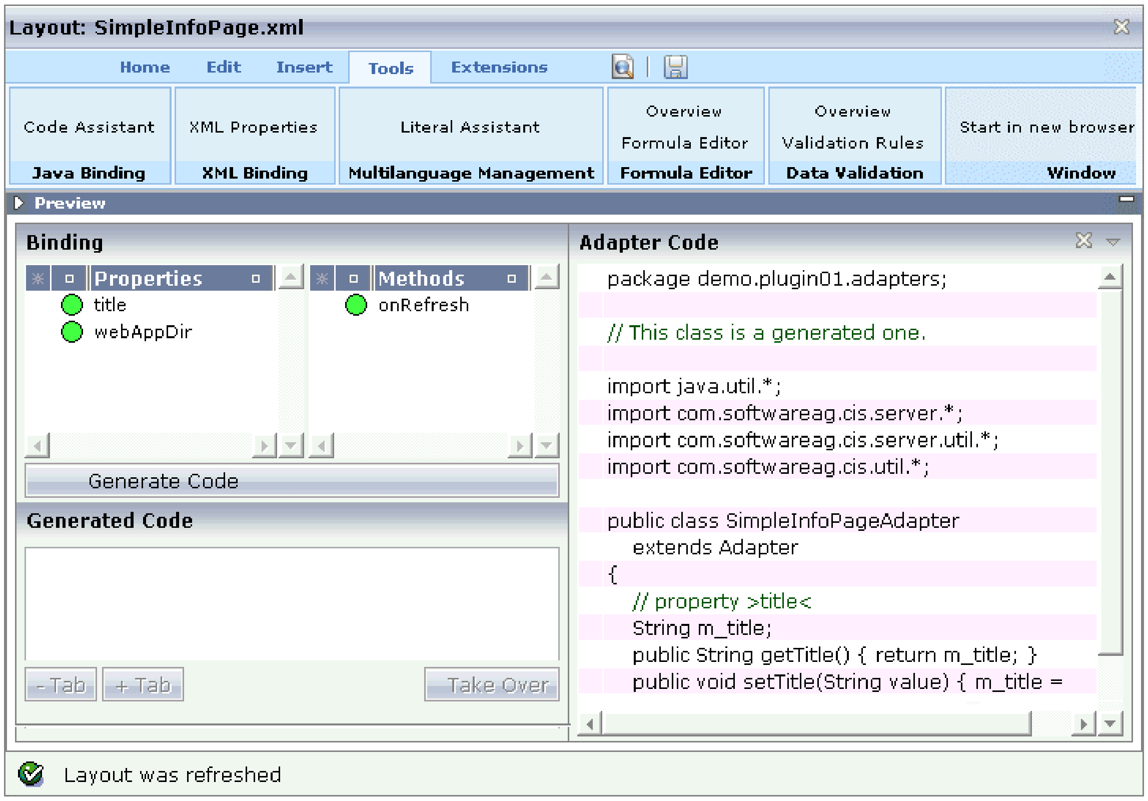
Name	ciseditorconfig.xml
Directory	C:/Documents and Settings/All Users/Application Data/Software AG/Tomcat/cp8.0/webapps/PluggableUI/DemoPlugIn01

Code Assistant Settings

General

Source Directory	AG/Tomcat/cp8.0/webapps/PluggableUI/DemoPlugIn01/src
Spaces Per Tab	4
Property Method Within One Line	✓

3. Click the **Save and Apply** button at the top of the dialog.
4. In the **Layout** view select the topmost tree node called **page** (you probably need to scroll up).
5. Change the `model` property of **page** from `DummyAdapter` to `demo.plugin01.adapters.SimpleInfoPageAdapter`.
6. Save the layout (using the diskette symbol). The content of the **Preview** view changes and indicates an error now. This is normal and can be ignored at the moment.
7. Switch to the **Tools** tab of the IDE and select **Code Assistant**. The look of the IDE changes and the generated code for our page adapter is visible on the right side.



```
package demo.plugin01.adapters;

// This class is a generated one.

import java.util.*;
import com.softwareag.cis.server.*;
import com.softwareag.cis.server.util.*;
import com.softwareag.cis.util.*;

public class SimpleInfoPageAdapter
    extends Adapter
{
    // property >title<
    String m_title;
    public String getTitle() { return m_title; }
    public void setTitle(String value) { m_title =
```

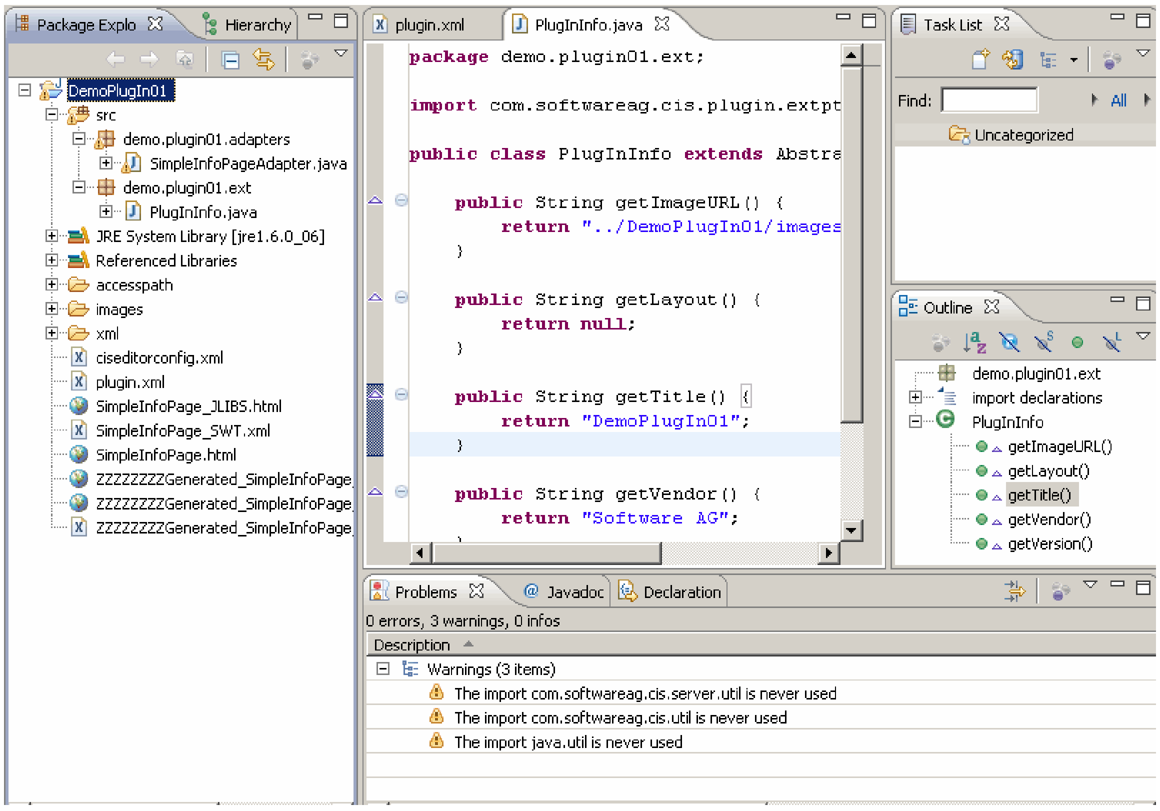
Layout was refreshed

We could apply the necessary changes for the adapter class using the IDE. The more convenient way is doing this inside of our already existing eclipse project to which we will switch back soon. One more step inside the IDE is missing: the code is not yet stored in the file system. Hence, press the diskette icon again! Now the adapter source code is stored in the Java source directory of our CentraSite Control plug-in DemoPlugIn01.

You can close the IDE now.

Now return to your Eclipse environment. We need to refresh our plug-in project. To do so, select the folder DemoPlugIn01 and choose **Refresh** from the context menu. After doing this and expanding all folders that relate to our plug-in, your eclipse project should look like the one below. Please note the contents of subfolders accesspath and xml which were formerly empty. The contents have been created by our IDE activities. Most importantly, you should notice that there is now a new package called demo.plugin01.adapters containing the class SimpleInfoPageAdapter.

Note that 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.



```
package demo.plugin01.ext;

import com.softwareag.cis.plugin.extpt

public class PlugInInfo extends Abstra

    public String getImageURL() {
        return "../DemoPlugIn01/images
    }

    public String getLayout() {
        return null;
    }

    public String getTitle() {
        return "DemoPlugIn01";
    }

    public String getVendor() {
        return "Software AG";
```

0 errors, 3 warnings, 0 infos

Warnings (3 items)

- The import com.softwareag.cis.server.util is never used
- The import com.softwareag.cis.util is never used
- The import java.util is never used

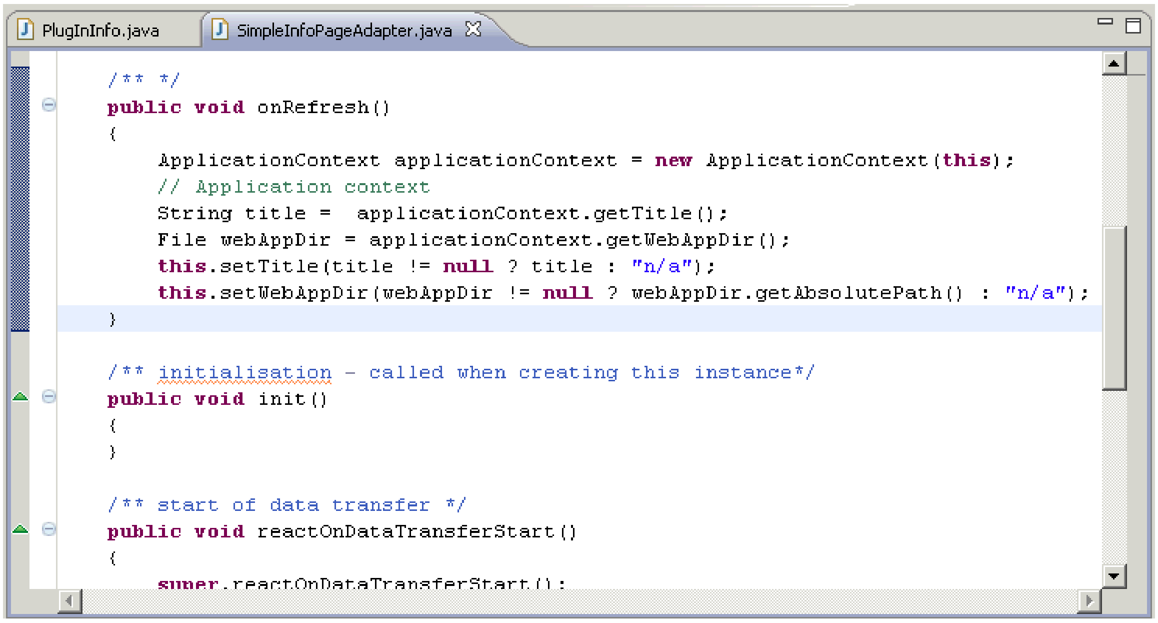
Let's apply application code to the adapter SimpleInfoPageAdapter now. To do so, open file SimpleInfoPageAdapter.java by double-clicking its node in the tree, and enter the following code inside the body of method onRefresh:

```
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 will be marked by the Java editor as unknown when you enter the code. So please press <Ctrl>+<Shift>+O to instruct the Java editor to add the necessary import statements automatically (for the missing class called File please choose java.io.File from the resulting dialog).

Now save the file. There should be no compilation errors.



```
    /** */
    public void 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");
    }

    /** initialisation - called when creating this instance*/
    public void init()
    {
    }

    /** start of data transfer */
    public void reactOnDataTransferStart()
    {
        super.reactOnDataTransferStart();
```

Now the core of this step: we will bring our user-defined layout inside CentraSite Control to the screen. The questions here are when and how we do this. The first (when) is easy to answer: on user request. For the second (how) there are a lot of possibilities. Using an extension point defined by CentraSite Control suggests itself. But which one do we choose?

In this tutorial step we will extend CentraSite Control at another point in order to add a perspective. Perspectives are listed on the top of the workbench. Once again we have to inform the pluggable infrastructure that we are extending CentraSite Control at a new point. In order to do so, add the following XML element to the plug-in description file plugin.xml in your Eclipse environment and save it afterwards.

Add the following `requiredPlugin` XML element after the existing `requiredPlugin` XML element:

```
<requiredPlugin id="com.centrasite.control" />
```

Also 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>
```

Save the file plugin.xml.

The implementation of our new perspective requires a new class which implements the necessary interface:

1. 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.
2. Let method getTitle() return the string `DemoPlugIn01`.
3. Let the getLogoImageURL() method return the path to our 24x24 icon (../DemoPlugIn01/images/star-24x24.gif).
4. Insert the methods

   ```
   public boolean hasTopicTree()
   {
    return false;
   }
   ```

   and

   ```
   public boolean supportsViews()
   {
    return false;
   }
   ```

   The Java source should look exactly like this then:

   ```
   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)
    {
    }

  }
```

5. Save and close the Java source file.

We will now extend CentraSite Control with a new topic. In order to do so, add the following XML element to the plug-in description file plugin.xml in your Eclipse environment and save it afterwards.

```
<!-- Topic -->
<extension point="com.softwareag.cis.plugin.topic"
  id="DemoPlugIn01Topic"
  perspective="demo.plugin01.DemoPlugIn01Perspective"
  class="demo.plugin01.ext.PlugInTopic" >
</extension>
```

The implementation of our new topic requires two new classes: the topic class itself which implements the necessary interface for the extension point and an adapter class for the topic. Let's start with the implementation of the adapter class:

1. 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.
2. Add a public default constructor to the class. The Java source should look exactly like this then:

   ```
   package demo.plugin01.ext.adapters;
   ```

```
import com.centrasite.control.adapters.TopicAdapter;

public class PlugInTopicAdapter extends TopicAdapter

{
   public PlugInTopicAdapter()
   {
   }

}
```

3. Save and close the Java source file.

And now the extending class:

1. 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. Check the box labeled **Inherited abstract methods**.
2. Add a public default constructor which invokes the super(int) constructor to the source:

   ```
   public PlugInTopic ()
   {
      super(0);
   }
   ```

3. Let the method getTopicAdapterClass() return PlugInTopicAdapter.class.
4. Let method getTitle() return the string `DemoPlugIn01`.
5. 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);
   ```

   After this change of the source code you should press <Ctrl>+<Shift>+O to resolve compilation problems.

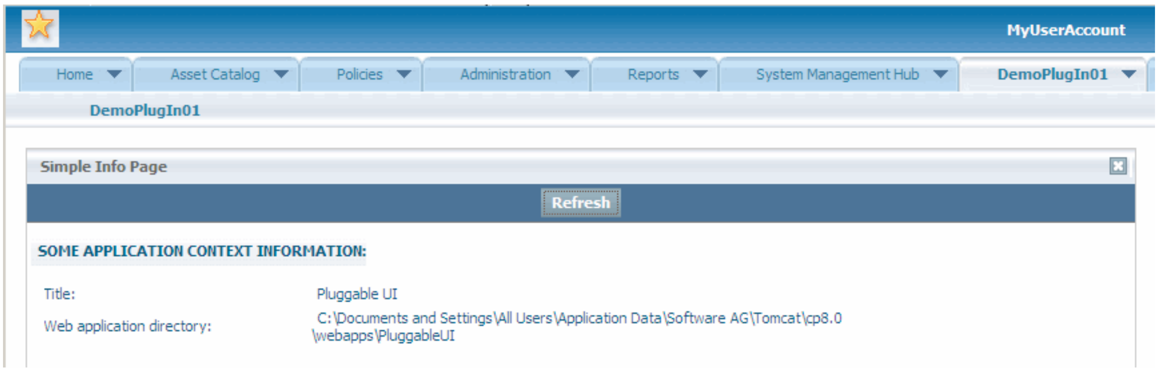
6. Save the Java source file and make sure that no compilation errors occur. After applying the changes described above, the Java source code for class PlugInTopic should look like this:

   ```
   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;
    }

  }
```

To see how our new extension affects CentraSite Control, restart the Software AG Runtime service and open CentraSite Control afterwards. The navigation pane shows the new perspective **DemoPlugIn01** which has 1 topic entry called **DemoPlugIn01**. Note also that the star-24x24.gif graphic is visible in the header bar.

When you click **Refresh** in the **Simple Info Page** display, the values for **Title** and **Web application directory** are updated:



SOME APPLICATION CONTEXT INFORMATION:

Title:	Pluggable UI
Web application directory:	C:\Documents and Settings\All Users\Application Data\Software AG\Tomcat\cp8.0 \webapps\PluggableUI

5 Application Framework

- Introduction 218
- Configuration 221
- Mapping Beans to Registry Objects with Annotations 223
- Querying the Registry 234
- Event Mechanism 243
- Asset Types 243
- Association Types 245
- Lifecycle Management 246
- Revision Management 247
- Multi-User Scenarios 249
- Setting the Classpath 250
- Examples 250

Introduction

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 UI is built.

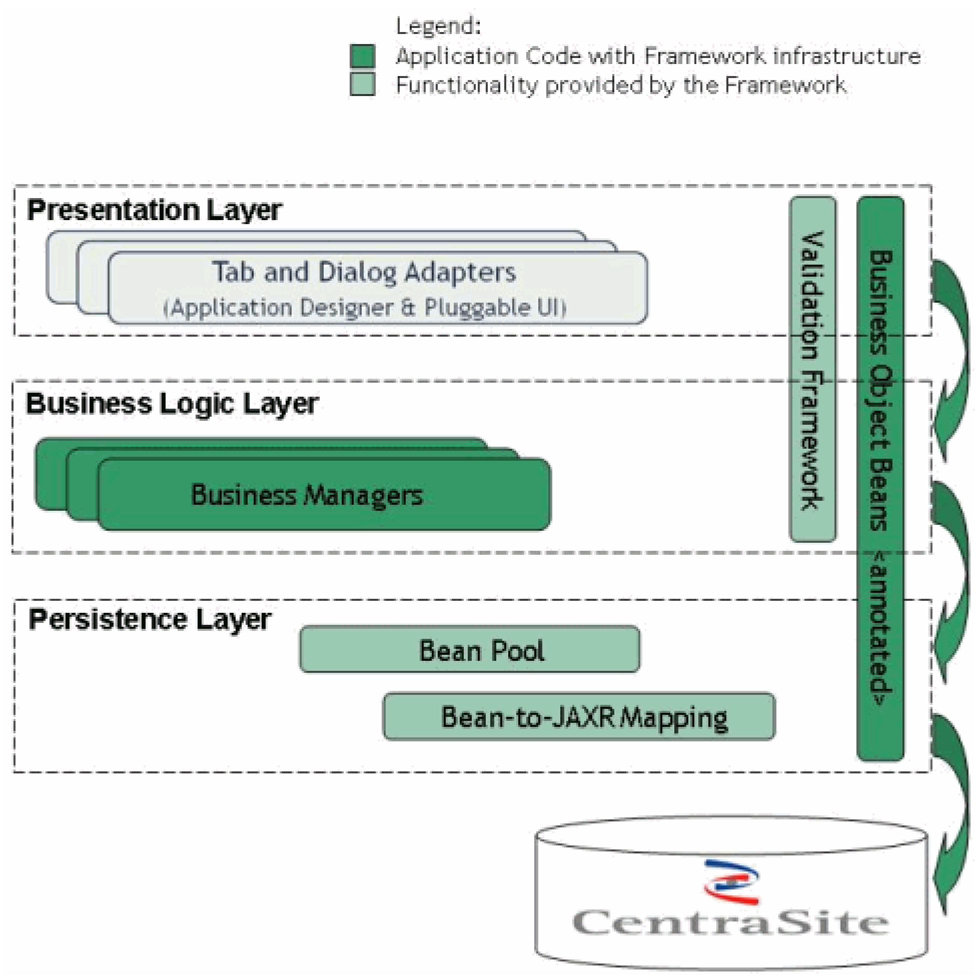
It contains two independent parts: the persistence framework and the validation framework.

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. It 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, i.e., constraints apply only when specific conditions about the bean are met.



This figure above shows 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.

The following topics are discussed in this topic:

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 fully qualified URL for the CentraSite registry/ repository. Default value is `http://localhost:53307`
user	The user ID of the CentraSite user.
password	The password of the user identified by the parameter – `user`.

browserBehaviour	Sets the `com.centrasite.jaxr.BrowserBehaviour` property of the connection factory. To enable type management, this flag must be set to `true`; to enable RevisionManagement it must be `false`. Default value is `false`.

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();
```

Configuration

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 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.

The following topics are discussed in this topic:

Bean Types Managed by CSAF

The framework keeps an internal data model for user-defined bean classes, i.e., 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, e.g., 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 will automatically be re-read from the registry, i.e., 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 that if this feature is turned on the current client of CSAF will override any changes made by another client.

Mapping Beans to Registry Objects with Annotations

Introduction to Bean Mapping

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 table below describes the annotations currently supported by the CentraSite Application framework.

Annotations and Description	Scope	Properties
@RegistryObject Maps a bean to a registry object with a specific object type.	Type	`objectTypeName` (optional) – the name of the object type of the registry object. `objectTypeKey` (optional) – the key of the object type. At least one of the properties must be specified.
@Property 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.	Method	`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.
@Slot 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.	Method	`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. `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. `type` (optional) – the type of the bean property. Supported types are BOOLEAN, DATE, CALENDAR, TIMESTAMP, INTEGER and AUTO. The latter allows the mapper to guess the property type.
@Slots	Method	`targetType` (mandatory) - the type of bean that is to be mapped to a single slot.

Annotations and Description	Scope	Properties
Maps all slots of a registry object to a bean property (Collection).		
@SlotProperty 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.	Method	`target` (mandatory) – can be one value from the enum SlotPropertyName – NAME, SLOT_TYPE, VALUES.
@TelephoneNumbers Maps a bean property to the TelephoneNumber object from the JAXR-based infomodel. Such objects are used in the User JAXR Object.	Method	`type` (optional) – the type of the telephone numbers.
@ExternalLink Maps a bean property to a ExternalLink JAXR-based object or a collection of them.	Method	`slotName` (optional) – the name of a slot inside the ExternalLink registry object to be mapped that is checked for having a specified value. This is used to pick the proper ExternalLink if the registry object has more than one. `slotValue` (optional) – the value of the slot to be checked. `type` (optional) – type of the bean used for the mapping
@Association Maps a property to an association. It can be either the association object itself or the target of the association.	Method	`key` (optional) – the key of the association type to be used. Either type or key must be present. `type` (optional) – the association type to be used. Either type or key must be present.

Annotations and Description	Scope	Properties
		`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. `mappedTo` (optional) – the property can be mapped to either the association registry object or the target of the association. `cascadeStyle` (optional) – Supported cascade styles are ALL (Cascade on all operations), UPDATE (Cascade on update operations), DELETE (Cascade on delete operations), NONE (no cascading).
@AssociationTarget 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.	Method	None
@Classification 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) or enumeration – Bean property <-> Classification (Concept) which concept is under a specified parent concept. The latter provides	Method	`classificationScheme` (optional) – the name of the ClassificationScheme to be used. `parentConcept` (optional) – the path of the parent concept. Used when mapping enumeration classifications. `parentConceptKey` (optional) – the key of the parent concept. Either the path or the key can be used. `conceptPath` (optional) – the path of the concept for this classification.

Annotations and Description	Scope	Properties
a set of predefined possible concepts, thus is similar to the notion of enumeration.		`conceptKey` (optional) – the key of the concept for this classification. Either the path or the key can be used. `targetType` (optional) – the type of the bean used for the mapping. Required when the property is a collection and the type cannot be guessed. `mappedTo` (optional) – the bean can be mapped either to the classification object or to its concept. `cascadeStyle` (optional) – The supported cascade styles are ALL, UPDATE, DELETE and NONE.
@ClassificationConcept Used in conjunction with the @Classification annotation. Maps a bean to the Concept of the Classification specified in the @Classification annotation.	Method	None
@ClassifiedInstances 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.	Type	`instances` (mandatory) – the array of the instances that this mapping will address.
@ClassifiedInstance Sets the information for a specific mapping between a bean from the hierarchy and a registry object.	Type	`classificationScheme` (mandatory) – the classification scheme to which the concept belongs. Either the scheme name or the key must be specified.

Annotations and Description	Scope	Properties
		`classificationSchemeKey` (mandatory) – the key of the classification scheme. Either the scheme name or the key must be specified. `conceptKey` (mandatory) – the key of the concept used to classify this instance. `conceptPath` (mandatory) – the path of the concept used to classify this instance. `beanType` (mandatory) – the type of the bean that corresponds to this instance.
@ClassificationAttribute 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 {@link Classification} 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).	Method	`attributeName` (mandatory) – The name of the attribute represented by this annotation. `cascadeStype` (optional) – The cascading style for this mapping. `targetType` (optional) – The type of the mapped bean. The bean itself must be of type Concept.
@FileAttribute Annotation for mapping the return value of a	Method	`attributeName` (mandatory) – The name of the attribute represented by this annotation.

Annotations and Description	Scope	Properties
(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 {@link ExternalLink} 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).		`cascadeStype` (optional) – The cascading style for this mapping. `targetType` (optional) – The type of the mapped bean. The bean itself must be of type ExternalLink.
@Relationship 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).	Method	`attributeName` (mandatory) – The name of the attribute represented by this annotation. `cascadeStype` (optional) – The cascading style for this mapping. `targetType` (optional) – The type of the mapped bean. The bean itself must be of type Concept.

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.ServiceIn
terfaceVersionImpl") 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,
                 cascadeStype = 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 from the Command Line

You can use the command line interface `CentraSiteCommand.cmd` (Windows) or `CentraSiteCommand.sh` (UNIX) of CentraSite. The command line tool is located in the directory *<CentraSiteInstallDir>* /utilities.

Use the `GenerateCSAFBeans` to generate registry beans in CentraSite.

The syntax for the command is:

```
C:\SoftwareAG\CentraSite\utilities>GenerateCSAFBeans.cmd <USERNAME>
<PASSWORD> <CENTRASITE-URL> <TYPENAME> <INTERFACEPACKAGE> <IMPLPACKAGE>
<DESTINATION>
```

The input parameters are:

Parameter	Description
USERNAME	The user ID of a registered CentraSite user. For example, a user who has the CentraSite Administrator role.
PASSWORD	The password for the registered CentraSite user identified by the parameter `-user`.
CENTRASITE-URL	**(Optional)** The URL of the CentraSite registry. Default value is `http:/localhost:53307`.
TYPENAME	*Required*. The namespace or name of the type to be generated. Example: {http://test}TestService. Or, the name of the virtual type to be generated. Example: "Virtual service" . **Note:** The quotation marks are necessary, in order that "Virtual service" is parsed as a single token.

Parameter	Description
INTERFACEPACKAGE	*Required*. The name of the package in which the interfaces should be generated. For example: `com.sag.generated`
IMPLPACKAGE	*Required*. The name of the package in which the implementation should be generated. For example: `com.sag.generated.impl`
DESTINATION	*Required*. The location where the generated bean will be stored.

Example:

```
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 222.

- **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 bean> beanClass)
  ```

  The search object searches the registry only for items from the specified type.

The search object has a result() method which 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

method name	The comparison operator:
	and — logical conjunction
	eq — equal

ge	greater than or equal to
gt	greater than
le	less than or equal to
like	matches a string that can include wildcards
lt	less than
ne	not equal
or	logical disjunction

property name

The name of the property to be compared. This property name is a string value representing the name of the Java property (`getName()` corresponds to “name”). The search functionality supports adding a sequence of properties. This is accomplished by knowing the searched RegistryBean property hierarchy and by separating following properties with a dot `.`.

Example:

Predicates.*eq*("externalLink.uri ", value)

The predicate is created for the URI property of the externalLinks of the searched RegistryBean, which should be equal to the given value.

value

The value to compare against. Most methods expect an Object value because the search can handle a variety of objects including String, Number, Date, Calendar, Key, RegistryBean and others. There are also methods that expect a specific value type. An example is like (String propertyName, String value), which supports wildcards and therefore the 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 will be 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 will fail. 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 will be searched and the ones that have name equal to “somePropertyname2” or ExternalLink with URI equal to “http://www.softwareag.com” will be 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 addAnnnotationProcessor(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. Below is a 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, and 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/or 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.deleteAsssociationType(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 LCMAdminManager
LCMAdminManager lcmAdminManager = sessionContext.getLCMAdminManager();

// 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 LCMAdminManager
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 etc. 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 should 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, etc.) 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.LockNotAvailableExcep
  tion
  ```
- 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.ObjectOutdatedExcept
  ion
  ```

  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 should minimize the time a registry object is kept locked in the database, i.e., 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.

6 API for JAXR

- Introduction to the CentraSite API for JAXR 252
- Creating and Closing a JAXR-based Connection 252
- Defining a Service 255
- Service that Uses Another Service 255
- Service with Additional Information 256
- Pre-Defined Classification Schemes (Taxonomies) 256
- Impact Analysis 258
- CentraSite API for JAXR Reference Information 258

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 via the XQJ-based (XQuery API for Java) interface.

Creating and Closing a JAXR-based Connection

Creating a JAXR-based Connection

To create a JAXR-based connection

1. Ensure that the CLASSPATH includes directories that contain the following files:

   activation.jar
   CentraSiteCommons.jar
   CentraSiteDynLoader.jar
   CentraSiteJAXR-API.jar
   CentraSiteLCM.jar
   CentraSiteLCM-api.jar
   CentraSiteLCM-L10N.jar
   CentraSitePolicy-API.jar
   CentraSiteResourceAccess-API.jar
   CentraSiteUtils.jar
   CentraSiteUtils-L10N.jar
   CentraSiteVMS.jar
   CentraSiteVMS-L10N.jar
   commons-codec.jar
   commons-httpclient.jar

commons-lang.jar
commons-logging.jar
cstUtils.jar
groovy-all*.jar
inmUtil.jar
inmUtilConf.jar
jaxen.jar
jaxr-api.jar
jaxrpc.jar
jdom.jar
log4j.jar
PolicyLogBindings.jar
saaj.jar
saxpath.jar
script-api.jar
sin-common.jar
sin-misc.jar
sin-ssx.jar
sin-xmlserver.jar
stax-api.jar
TaminoAPI4J.jar
TaminoAPI4J-l10n.jar
uddiKeyConverter.jar
wstx-asl.jar
wvcm.jar
xmlbeans.jar
xqjapi.jar
xqj-ino-api.jar

Note: You can find these files in the CentraSiteredist 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. Start your client program with the following parameter:

```
-Djavax.xml.registry.ConnectionFactoryClass=com.centrasite.jaxr.Connectio
nFactoryImpl
```

   - Or:

     Set this property during program startup:

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

3. Create a factory:

```
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.

Note: 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 more often. After an asset is read, it will be 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. With the connection given, the other environment objects can easily be constructed:

   ```
   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. We therefore strongly recommend making sure that a connection is closed in case of a JAXR-based client failure. Otherwise the resources are released only after a non-activity timeout; this might hinder parallel users.

To close a JAXR-based connection

1. Use the following:

   `connection.close();` where connection is as specified in the example above.

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.findClassificationSchemeByNam
e(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 comes with 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 itself 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 `ContentType`. This is an external classification scheme.

- 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

Impact Analysis

Impact analysis means finding dependencies between objects: which object depends on which other object, or vice-versa: if one object is modified or deleted, which other objects are affected? For example, if a web service interface changes, which callers must be adapted?

In JAXR, dependencies between objects are established via associations. There are a variety of pre-defined association types, and moreover a JAXR-based client can create its own association types. Although the names of the association types - for example HasChild or HasMember - suggest a certain semantic, JAXR itself does not imply any semantics with the association types. CentraSite supports the following conventions for associations.

If there is a dependency between two objects, each of which can exist on its own, then this dependency should be expressed by a Uses association. Example: one web service calls another web service. Remember that JAXR-based associations are directed: the association's source object should be the caller/user (in general, the object that depends on another object), and the association's target is the called/used object.

If there is an object `C` that cannot exist without another object `P`, then `C` should have a HasParent association to `P`. Example: A table object cannot exist without a database object, hence there is a HasParent association from each table to the corresponding database.

The reason for preferring HasParent over the “inverse” HasChild association is as follows: CentraSite tries to maintain referential integrity; this means, among other things, that is not possible to delete an object that is still the target of an association. Hence associations should be directed in such a way that an object cannot be deleted if someone else still depends on it: an object should not be deleted if it still has children, or if it is still in use by someone else.

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 itself also uses XQJ to access the registry.

The CentraSiteCentraSiteConnection 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 he already has a JAXR-based connection:

```
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 he has 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, he must 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() throws 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 lost updates. 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 his or her 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. Bear in mind that in order to modify an object, the JAXR-based client always has to read it first, then modify the Java instance, then call saveObjects() in order to write the modified object back to the database.

Scenario A

JAXR-based Client A	JAXR-based Client B
1. Read a RegistryObject.	

JAXR-based Client A	JAXR-based Client B
	2. Read the same RegistryObject.
3. Start to modify the object. This automatically locks the object.	
	4. Start to modify the object. The attempt to lock the object fails and a LockNotAvailableException is thrown.

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	JAXR-based Client B
1. Read a RegistryObject.	
	2. Read the same RegistryObject.
3. Start to modify the object. This automatically locks the object.	
4. Save the object. This releases the lock.	
	5. Start to modify the object. The attempt to lock the object fails and an ObjectOutdatedException is thrown.

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 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 is thrown. This ensures that updates are not lost and that all modifications are based on the latest state of the object.

Immediately before the ObjectOutdatedException is thrown, the API cleans up its internal structures. When the client catches 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, it is created. This new user object is not added to any organization.

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 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 because 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 delinked 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, i.e., it has not yet been saved, or
- it is the target of an association.

Deleting a registry object has the following side-effects:

1. Remove the object from all its packages; update the packages.
2. Delink the object from all its external links; update the external links.
3. Delete all associations whose source object is the object to be deleted.
4. Delete all classifications whose classified object is the object to be deleted.
5. Delete 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; otherwise:
2. Delete the classification scheme.

Concept

1. Reject deletion if there are child concepts; otherwise:
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; otherwise:
2. Delete the external link.

Organization

1. Reject deletion if there are child organizations, services, or users; otherwise:
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; otherwise:
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 throw an UnsupportedCapabilityException exception:

- 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 %”, i.e. 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. We recommend specifying the `slotType` as an `xs:string`.

Caching Considerations

This topic describes the following 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 will also be in the cache, and the call will return 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 will be 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, i.e. Control 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 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 choosing the **Refresh** button.

Dynamically Loaded JAR Files

The system 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 also that problems may arise if a custom security manager has been implemented, because the connection to the database will be refused.

Cache Location

The system 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`.

7 API for XQJ

- Introduction to the API for XQJ 270
- What is XQJ? 270
- Working with the XQJ Interface 271
- CentraSite-Specific Extensions to XQJ 275
- XQDataSource Properties 276

Introduction to the API for XQJ

You can use XQJ, the XQuery API for Java™, for processing XML and for data integration applications. This chapter introduces the CentraSite implementation of XQJ and its features. It explains how to use the API and provides examples for each type of task.

The reader of the document should be an experienced Java programmer.

What is XQJ?

XQJ, the XQuery API for Java, is based on XQuery, a query language promulgated by the W3C that can operate both on physical XML documents, and also on virtual XML documents that have been derived from data sources such as relational or object databases. XQJ is a powerful new API standard developed for invoking XQuery expressions against virtually any XML or relational database and processing query results. XQJ makes the full power of the XQuery language available to Java applications. You can programmatically process the results in your Java code in a JDBC-like manner. XQJ is to XQuery what JDBC is to SQL.

The XQJ standard specifies a number of Java interfaces. The CentraSite XQJ interface implements the functionality defined by these interfaces, and thus makes XQJ available to the application; in addition, the CentraSite XQJ interface implements extensions that support CentraSite-specific features.

Note: Beginning with version 8.2, CentraSite supports the final release of the XQJ specification (in contrast, earlier versions of CentraSite supported a preliminary release of the XQJ specification). Note that the XQJ interface that is implemented by current versions of CentraSite is not compatible with the interface that was implemented by versions of CentraSite prior to version 8.2. Documentation of the prior XQJ interface is available to Software AG customers who have a current maintenance contract in Empower.

Features of the XQJ Interface

The CentraSite XQJ interface supports:

- Prepared XQueries
- The submission of queries to the CentraSite registry/repository
- XQuery updates
- Transaction control (commit, rollback)
- User authentication prior to connecting to the database

- Variable binding to parameterize queries
- Handling registry/repository errors and warnings
- The creation and execution of materialized sequences and items
- Different models for accessing data in the CentraSite registry/repository (DOM, SAX, and StAX-compatible streams)

Working with the XQJ Interface

If you want to develop an XQJ application, you will find the classes of the CentraSite XQJ implementation in the jar file rts/bin/xqj.jar under the CentraSite installation location.

You can use the CentraSite XQJ interface to perform an XQuery on the basis of a standard XQExpression or an XQPreparedExpression. With a standard XQExpression, the query is parsed each time it is executed. If a query is to be executed many times, it can be more efficient to use an XQPreparedExpression, which is parsed only once.

Executing an XQuery with a Standard XQExpression

To execute an XQuery with a standard XQExpression

1. Invoke the getXQConnection() method to get the XQConnection object from the JAXRConnection.

   Example

   ```
   /* Get the XQConnection from the JAXRConnection */
   XQConnection connection = jaxrConnection.getXQConnection ();
   ```

   You have now established an XQConnection.

2. Create an XQExpression object from the XQConnection object. The XQExpression is used to invoke several other methods to perform various tasks using the CentraSite XQJ interface. You may create more than one XQExpression from a single connection if required.

   Example

   ```
   /* Create XQExpression from XQConnection to execute an XQuery. */
   XQExpression expression = connection.createExpression();
   ```

3. Optionally, you can bind one or more external variables. An external variable is a type of variable that can be dynamically added to the query by declaring the variable in the query. The value of the variable can be set externally and added to the pre-set variable while executing the XQuery.

   Example

   ```
   String xquery = "declare variable $year as xs:int external" +
       "for $q in input()/bib/book where $q/@year > $year return $q" ;
   XQExpression expression = connection.createExpression();
   ```

```
expression.bindInt(new QName("year"),
    1993,XQItemTypeHelper.createIntXQItemType());
XQResultSequence xqResultSequence = expression.executeQuery(xquery);
```

4. Invoke the executeQuery() method. This returns an XQResultSequence.

   Example

```
/* Executing an XQuery */
/* Instance of the query string: */
String xquery = "for $b in input()/book return $b/title";
/* Execute the above XQuery String, which returns an XQResultSequence */
XQResultSequence xqResultSequence = expression.executeQuery(xquery);
```

5. The XQResultSequence represents the XQuery result. Retrieve the query result and read/print it in XML format. The query result sequence is displayed item by item. Using XQJ, it is possible to get the result sequence in DOM, SAX and StAX-compatible formats.

   Note: You cannot scroll the XQResultSequences backwards.

   Example

```
/* Iterating the XQResultSequence */
XMLStreamReader reader = null;
While(xqResultSequence.next())
{
 reader =  xqResultSequence.getItemAsStream();
 /* Iterate the XML StreamReader using StAX-compatible APIs */
}
connection.commit();
connection.close();
```

   Example using the getInt() method

```
/* Instance of the XQuery String */
String xquery =" for $b in input()/bib/book return xs:int($b/@year) ";

/* This query on execution will return the year as an integer value */
XQResultSequence xqResultSequence = expression.executeQuery(xquery);

xqResultSequence.next();
int I = xqResultSequence.getInt();
```

   Example using the getAtomicValue() method

```
/* Instance of the XQuery String */
String xquery = "for $p in input()/book return xs:string($p/title)";
XQResultSequence xqResultSequence = expression.executeQuery(xquery);

/* This query on execution will return the title as a String */
xqResultSequence.next();
String str = xqResultSequence.getAtomicValue();
```

   Example using the getNode() method

```
/* Instance of the XQuery String */
String xquery = "for $q in input()/bib/book return $q";
XQResultSequence xqResultSequence = expression.executeQuery(xquery);
xqResultSequence.next();
Node node = result.getNode();
```

   Example using the writeItemToSAX() method

```
xqResultSequence.next();
StringWriter sw = new StringWriter();

/* Provide a org.xml.sax.ContentHandler, which is saxhandler   */
/* in our case                                                 */
XQSAXTextEventHandler saxhandler = new XQSAXTextEventHandler(sw);
resultSequence.writeItemToSAX(saxhandler);
System.out.println(sw);
```

6. Finally, invoke the XQConnection.close() method to close the connection to the registry/ repository.

   Example

   ```
   /* Commit and close the XQConnection once you have completed    */
   /* working with it                                              */
   connection.commit();
   connection.close();
   ```

Executing an XQuery with an XQPreparedExpression

To execute an XQuery with an XQPreparedExpression

1. Invoke the getXQConnection() method to get the XQConnection object from the JAXRConnection.

   Example

   ```
   /* Get the XQConnection from the JAXRConnection */
   XQConnection connection = jaxrConnection.getXQConnection ();
   ```

   You have now established an XQConnection.

2. Create an XQPreparedExpression object from the XQConnection object. The XQPreparedExpression is used to invoke several other methods to perform various tasks using the CentraSite XQJ interface. You may create more than one XQPreparedExpression from a single connection if required.

   Example

   ```
   /* Create XQPreparedExpression from XQConnection */
   String pQuery = "for $q in input()/bib/book return $q";
   XQPreparedExpression preparedExpression = conn.prepareExpression(pQuery);
   ```

3. Optionally, you can bind one or more external variables. An external variable is a type of variable that can be dynamically added to the query by declaring the variable in the query. The value of the variable can be set externally and added to the pre-set variable while executing the XQuery.

   Example

   ```
   /* Binding variables in Prepared Expressions */
   String pQuery = "declare variable $int as xs:int external" +
        "for $q in input()/bib/book where $q/@year = $int return $q";

   XQPreparedExpression preparedExpression = conn.prepareExpression(pQuery);

   /* Bind the appropriate value to the prepared expression */
   /* using the matching binding API provided. */
   ```

Using bindInt() to bind an int value to the prepared expression

```
preparedExpression.bindInt(new QName("int"),
   1994, XQItemTypeHelper.createIntXQItemType());
```

Using bindNode() to bind a node to the prepared expression

```
/* Get a node to bind by executing an expression */
XQExpression expression = connection.createExpression();
XQResultSequence xqResultSequence =
   expression.executeQuery("for $q in input()/bib/book
       where $q/@year = 1994 return $q/title");
xqResultSequence.next();

/* Get a node from the result sequence retrieved above */
Node node = xqResultSequence.getNode();

/* PreparedQuery */
String pquery = "declare variable $node external " +
    "for $q in input()/bib/book where $q/title = $node return $q";
XQPreparedExpression prepared = connection.prepareExpression(pQuery);

/* Bind the above retrieved node to the prepared query */
prepared.bindNode(new QName("node"), node);
```

4. Invoke the executeQuery() method. This returns an XQResultSequence.

   Example

```
/* Execute the prepared expression which returns an XQResultSequence */
XQResultSequence xqResultSequence = preparedExpression.executeQuery();
```

5. The XQResultSequence represents the XQuery result. Retrieve the query result and read/print it in XML format. The query result sequence is displayed item by item. Using XQJ, it is possible to get the result sequence in DOM, SAX and StAX-compatible formats.

   Example

```
/* Iterating the XQResultSequence */
XQResultSequence xqResultSequence = preparedExpression.executeQuery();
XMLStreamReader reader = null;
While(xqResultSequence.next())
{
 reader =  xqResultSequence.getItemAsStream();
 /* Iterate the XML StreamReader using StAX-compatible APIs */
}
```

Example using the getInt() method

```
XQResultSequence xqResultSequence = preparedExpression.executeQuery();
xqResultSequence.next();
int I = xqResultSequence.getInt();
```

Example using the getAtomicValue() method

```
XQResultSequence xqResultSequence = preparedExpression.executeQuery();
xqResultSequence.next();
String str = xqResultSequence.getAtomicValue();
```

Example using the getNode() method

```
XQResultSequence xqResultSequence = preparedExpression.executeQuery();
xqResultSequence.next();
Node node = result.getNode();
```

Example using the writeItemToSAX() method

```
XQResultSequence xqResultSequence = preparedExpression.executeQuery();
xqResultSequence.next();
StringWriter sw = new StringWriter();

/* Provide an org.xml.sax.ContentHandler, which is saxhandler in our case */
XQSAXTextEventHandler saxhandler = new XQSAXTextEventHandler(sw);
resultSequence.writeItemToSAX(saxhandler);
System.out.println(sw);
```

6. Finally, invoke the XQConnection.close() method to close the connection to the registry/ repository.

Example

```
/* Commit and close the XQConnection once you have completed working */
/* with it                                                           */
connection.commit();
connection.close();
```

Working with a Materialized XQSequence

A materialized sequence is not bound to any connection or XQuery expression. It can be created from XQResultSequences or from a java.util.iterator.

Examples

Creating a Sequence

This example demonstrates how to create a materialized sequence from Java collection via the java.util.iterator interface. It creates a materialized sequence holding 3 int items.

```
ArrayList items = new ArrayList();
items.add(conn.createItemFromInt(123,null));
items.add(conn.createItemFromInt(456,null));
items.add(conn.createItemFromInt(789,null));
XQSequence sequence = conn.createSequence(items.iterator());
```

Creating a Copy from an XQResultSequence

This example demonstrates how an XQResultSequence can be copied into a materialized sequence. The materialized sequence will exist independently of the XQResultSequence.

```
String query = "for $q in input()/bib/book where $q/@year = 1994 return $q";
XQExpression expression = connection.createExpression();
XQResultSequence resultSequence = expression.executeQuery(query);

XQSequence Sequence = connection.createSequence(resultSequence);
```

CentraSite-Specific Extensions to XQJ

CentraSite adds useful facilities to the XQJ interface for updating a database and for inserting a document into the CentraSite registry/repository.

Updating a Database Using XQJ

Using the CentraSite XQJ interface, you can update the registry/repository. This feature is a Software AG specific extension of XQJ.

To update an XQuery

1. Specify the string or the reader object containing the update XQuery
2. Invoke the executeUpdate() method on the expression.

```
String updateQuery =
"update for $q in input()/bib/book where $q/@year = 1994" +
"do replace $q/title with <title>XQJ from SoftwareAG </title>";
XQResultSequence xqResultSequence =
((XQExpressionImpl)expression).executeUpdate(updateQuery);
// execute update
```

Inserting a Document in the Registry/Repository

This feature is a Software AG specific extension of XQJ.

To insert a document in the

1. Specify the XML instance to be inserted as a string or the reader object.
2. Execute the executeInsert() method in XQExpression to insert a document.

```
String insertStr = "<your xml goes here>";
(XQExpressionImpl)expression).executeInsert(insertStr);
```

XQDataSource Properties

In addition to the standard properties, CentraSite offers the following properties for parameterizing XQJ connections. Note that user credentials, i.e. user-ID and password, are passed via standard properties, as shown below:

Standard Properties and Descriptions
`javax.xml.xquery.property.UserName` Unique user ID for connecting to the registry/repository.
`javax.xml.xquery.property.Password` The password for the specified user ID.
`javax.xml.xquery.property.MaxConnections`

Standard Properties and Descriptions
The maximum number of open connections that can be established from the datasource.

CentraSite-Specific Properties and Descriptions
`com.softwareag.tamino.xqj.dbUri` Mandatory. The URI of the database to which the user is connecting. This information is mandatory to connect to the datasource, which is the CentraSite registry/repository in this context.
`com.softwareag.tamino.xqj.defaultCollection` Mandatory. The name of the collection in the registry/repository that the user will access to query, update, or insert a document.
`com.softwareag.tamino.xql.locale` The locale to be set for the connection.
`com.softwareag.tamino.xqj.isolationLevel` Together with the `_lockMode` parameter, this parameter specifies the way in which two or more transactions in a session context can access the same data simultaneously. The isolation level can be set to `None`.
`com.softwareag.tamino.xqj.lockMode` Together with the `_isolationLevel` parameter, this parameter specifies the way in which two or more transactions in a session context can access the same data simultaneously.
`com.softwareag.tamino.xqj.lockWait` The action to be taken if data is not accessible to the current transaction because another transaction has used the `_isolationLevel` or `_lockMode` parameter to restrict access to the data.
`com.softwareag.tamino.xqj.fetchSize` The number of records to be retrieved at a time for display. This property accepts an integer value.
`com.softwareag.tamino.xqj.sensitive`

CentraSite-Specific Properties and Descriptions
The parameter `_sensitive` is required when opening a cursor with _xquery. Valid values are `no` and `vague`. If you specify `_sensitive=no`, an insensitive cursor is opened. This means that the query is calculated on a fixed input when the cursor is opened, and thus the result sequence remains unchanged as long as the cursor is active. If you specify `_sensitive=vague`, a vague cursor is opened. The query is calculated on an input that takes modification operations of parallel transactions into account. Thus, the result sequence can vary during the lifetime of the cursor if documents that match the original query criteria are inserted, updated or deleted in the meantime.
`com.softwareag.tamino.xqj.nonactivityTimeout` The non-activity timeout in seconds.

8 Java Management Interface

■ Introduction the Java Management Interface 280
■ Description 280
■ Attributes and Operations 281

Introduction the Java Management Interface

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

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

Description

Recent version of Java contain 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:

  <*CentraSiteInstallDir*>/rts/jmx/CentraSiteJMXAdmin.jar

  <*CentraSiteInstallDir*>/rts/jmx/CentraSiteJMXAdmin-L10N.jar

  <*CentraSiteInstallDir*>/rts/jmx/CentraSiteAdminAPI.jar

  <*CentraSiteInstallDir*>/rts/bin/inmUtil.jar

  <*CentraSiteInstallDir*>/rts/bin/inmUtilConf.jar

  <CentraSiteInstallDir>/rts/bin/log4j.jar

- The PATH must include the following directory
  - *<CentraSiteInstallDir>*/bin
- 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 topics are discussed in this topic:

Attributes

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

cache size
The size of the cache, in megabytes. If you change this attribute, the CentraSite Registry/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/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/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/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/Repository Start/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 via console software (e.g. a web browser), the operation is initiated when you select the **Invoke** button.

- Start the CentraSite Registry/Repository;
- Stop the CentraSite Registry/Repository in normal mode;
- Stop the CentraSite Registry/Repository in the specified mode;

  Possible termination modes are:

Code	Meaning
0 (normal)	Terminates the server session normally and waits for currently active processing to finish. The maximum waiting time (in seconds) can be set with the attribute `transaction timeout`.
1 (rollback)	Terminates the server immediately. User transactions that have not finished are rolled back.
2 (abort)	Terminates the server session immediately. All processing is stopped immediately. Crash dump files are written. Using this option initiates an automatic repair (autorepair) the next time the server is started. It should only be used as a last resort.

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

  Backs up the CentraSite Registry/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 will now be listed in the output of the `list all backups` operation. A backup is done to freeze the current state of the CentraSite Registry/Repository.
- Restore the contents of the CentraSite Registry/Repository from the latest backup with or without recovery.

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

  The `restore` operation can only be used when the CentraSite Registry/Repository is not active. It is used to restore the CentraSite Registry/Repository to the state that was stored in a previously-made 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 choose to 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/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.

9 Web Service Interfaces

■ Introduction to the Web Service Interfaces 286
■ Approval Service 286

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 (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:

- ImportWsdlService
- ImportXsdService
- ImportXPDLService
- ApprovalService

For information about importing objects using API, see the *CentraSite Administrator's Guide*.

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

Approval Service

About the 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

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 will return 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/repository whose approval log is to be queried. A client application must specify the `locationCentraSite` parameter if the registry/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/repository is running at its default location, a client is not required to specify the `locationCentraSite` parameter.

Retrieving the List of Approval Requests that a User Has Submitted

You 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* of the approval requests that a user has submitted. However, you can optionally set the `objectType`, `submittedAfter`, `submittedBefore`, and/or `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 (e.g., 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`, which indicates the total number of approval requests in the result set. If the operation did not find any approval requests that satisfied the operation's criteria, there will be no elements returned in `ApprovalRequest[]` and the `count` value will be 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" on page 290.

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 getApprovalActions 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 getApprovalActions 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, as follows
  - If the request has not yet been processed to completion (i.e., it has not yet been approved or rejected) and the approver has not taken any action on the request, the status will be "Pending".
  - If the approver has approved the request, the status will be "Approved".
  - If the approver has rejected the request, the status will be “Rejected”.
  - If the request has been processed to completion (i.e., it has been approved or rejected), approvers who did not make the approval decision will have the status "No Action". (If the approval request was auto-approved, all of the approvers will have the status "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, do the following:

1. Use the getPendingApprovals operation to obtain the list of requests that require the user's approval.
2. Apply the approve or reject 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 will contain the approval requests that were approved or rejected by the operation.

For more information about working with the contents of the ApprovalRequestList message, see "ApprovalRequestList Message" on page 288.

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 (e.g., 15 entries at a time). You might use this feature, for example, to display 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 or 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 (i.e., 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	`scroll.start` = 1	`scroll.number` = 20
getApprovalHistory again	`scroll.start` = 21	`scroll.number` = 20
getApprovalHistory again	`scroll.start` = 41	`scroll.number` = 20

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

To determine when 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 simply 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 become stuck in "pending" mode. To resolve this condition, you must back that request out of the approval system and resubmit it (after updating the approver group, of course).

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 that 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 will revert the object to state A and return 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.

Operations

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/or submission date.

For additional information about using this operation, see "Approving or Rejecting Approval Requests" on page 289.

Input Message

Parameter Name	Description
`locationCentraSite`	*String* Optional. The address of the CentraSite registry/ repository from which you want to retrieve the approval

Parameter Name	Description
	requests. The registry/repository runs at the following URL: http://*server:port* /CentraSite/CentraSite Where *server* is the machine on which the CentraSite registry/repository is running and *port* 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 `locationCentraSite`, the Approval service will use the following default URL: http://localhost:53307/CentraSite/CentraSite
`objectType`	*String Array* Optional. 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`	*DateTime* Optional. If you want to retrieve requests after a particular date, specify that date in this element.
`submittedBefore`	*DateTime* Optional. If you want to retrieve requests before a particular date, specify that date in this element.
`locale`	*String* Optional. The locale in which you want the results returned.
`scroll`	*Scroll* Optional. If you want to return a particular block of entries from the result set, specify the following values in the `scroll` element. For information about using the `scroll` element, see "Scrolling Through the List of Returned Approval Requests" on page 290.

`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`	*Integer* Optional. The number of entries to be returned in the block of approval requests.

Parameter Name	Description
	If you specify a `start` value, but no `number` value, the remainder of the result set is returned.

Output Message

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/or approval status.

For additional information about using this operation, see "Retrieving the List of Approval Requests that a User Has Submitted" on page 287.

Input Message

Parameter Name	Description
`locationCentraSite`	*String* Optional. The address of the CentraSite registry/ repository from which you want to retrieve the approval requests. The registry/repository runs at the following URL: http://*server:port* /CentraSite/CentraSite Where *server* is the machine on which the CentraSite registry/repository is running and *port* 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 `locationCentraSite`, the Approval service will use the following default URL: http://localhost:53307/CentraSite/CentraSite
`status`	*String* Optional. If you want to retrieve only requests with a specified approval status, specify one of the following values shown here: **Specify...** **To retrieve...**

Parameter Name	Description
	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`	*String Array* Optional. If you want to retrieve approval requests for only certain object types, specify the types by name in this element. **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`	*DateTime* Optional. If you want to retrieve requests after a particular date, specify that date in this element.
`submittedBefore`	*DateTime* Optional. If you want to retrieve requests before a particular date, specify that date in this element.
`locale`	*String* Optional. The locale in which you want the results returned.
`scroll`	*Scroll* Optional. If you want to return a particular block of entries from the result set, specify the following values in the `scroll` element. For information about using the `scroll` element, see "Scrolling Through the List of Returned Approval Requests" on page 290.
	`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`: *Integer* Optional. The number of entries to be returned in the block of approval requests.

Parameter Name	Description
	If you specify a `start` value, but no `number` value, the remainder of the result set is returned.

Output Message

ApprovalRequestList

getApprovalActions

This operation returns detailed information about specified approval requests.

For additional information about using this operation, see "Getting Details about the Actions of the Approvers Associated with a Request" on page 288.

Input Message

Parameter Name	Description
`locationCentraSite`	*String* Optional. The address of the CentraSite registry/ repository from which you want to retrieve the approval requests. The registry/repository runs at the following URL: http://*server:port* /CentraSite/CentraSite Where *server* is the machine on which the CentraSite registry/repository is running and *port* 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 `locationCentraSite`, the Approval service will use the following default URL: http://localhost:53307/CentraSite/CentraSite
`approvalRequestKeys`	*String Array* The keys for the approval requests whose details you want to retrieve.
`locale`	*String* Optional. The locale in which you want the results returned.

Output Message

ApprovalActionResult

approve

This operation approves specified approval requests.

For additional information about using this operation, see "Approving or Rejecting Approval Requests" on page 289.

Input Message

Parameter Name	Description
`locationCentraSite`	*String* Optional. The address of the CentraSite registry/ repository on which the approval requests reside. The registry/repository runs at the following URL: http://*server:port* /CentraSite/CentraSite Where *server* is the machine on which the CentraSite registry/repository is running and *port* 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 `locationCentraSite`, the Approval service will use the following default URL: http://localhost:53307/CentraSite/CentraSite
`approvalRequestKeys`	*String Array* The keys for the requests that are to be approved.
`comment`	*String* Optional. A comment from the approver.
`locale`	*String* Optional. The locale in which you want the results returned.

Output Message

ApprovalRequestList (will contain the requests that were approved)

reject

This operation rejects the specified approval requests.

For additional information about using this operation, see "Approving or Rejecting Approval Requests" on page 289.

Input Message

Parameter Name	Description
`locationCentraSite`	*String* Optional. The address of the CentraSite registry/repository on which the approval requests reside. The registry/repository runs at the following URL: http://*server:port* /CentraSite/CentraSite Where *server* is the machine on which the CentraSite registry/repository is running and *port* 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 `locationCentraSite`, the Approval service will use the following default URL: http://localhost:53307/CentraSite/CentraSite
`approvalRequestKeys`	*String Array* The keys for the requests that are to be rejected.
`comment`	*String* Optional. A comment from the approver.
`locale`	*String* Optional. The locale in which you want the requests returned.

Output Message

ApprovalRequestList (will contain 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 will receive all entries in the log. If the user belongs to the Organization Administrator role, he or she will receive 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
`locationCentraSite`	*String* Optional. The address of the CentraSite registry/ repository on which the approval history log resides. The registry/repository runs at the following URL: http://*server:port* /CentraSite/CentraSite Where *server* is the machine on which the CentraSite registry/repository is running and *port* 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 `locationCentraSite`, the Approval service will use the following default URL: http://localhost:53307/CentraSite/CentraSite
`status`	*String* Optional. If you want to retrieve only requests with a specified approval status, specify one of the values shown here:

Specify...	To retrieve...
In Progress	Approval requests that are pending (i.e., 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.

Parameter Name	Description
`objectType`	*String Array* Optional. If you want to retrieve approval requests for only certain object types, specify the types by name in this element. **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.

Parameter Name	Description
`submittedAfter`	*DateTime* Optional. If you want to retrieve requests after a particular date, specify that date in this element.
`submittedBefore`	*DateTime* Optional. If you want to retrieve requests before a particular date, specify that date in this element.
`locale`	*String* Optional. The locale in which you want the results returned.
`scroll`	*Scroll* Optional. If you want to return a specified block of entries from the result set, specify the following values in the `scroll` element. For information about using the `scroll` element, see "Scrolling Through the List of Returned Approval Requests" on page 290. `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` — *Integer* Optional. The number of entries to be returned in the block. If you specify a `start` value, but no `number` value, the remainder of the result set is returned.

Output Message

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.

For additional information about using this operation, see "Reverting a Pending Approval Request" on page 291.

Input Message

Parameter Name	Description
`locationCentraSite`	*String* Optional. The address of the CentraSite registry/ repository on which the object resides. The registry/ repository runs at the following URL: http://*server:port* /CentraSite/CentraSite Where *server* is the machine on which the CentraSite registry/repository is running and *port* 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 `locationCentraSite`, the Approval service will use the following default URL: http://localhost:53307/CentraSite/CentraSite
`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 `revertedState` parameter.

ApprovalRequestList

This data structure holds a list of approval requests.

For additional information about working with this structure, see "ApprovalRequestList Message" on page 288.

Parameter Name	Description
`ApprovalRequest[]`	An array of ApprovalRequest elements. Each ApprovalRequest entry in the array represents one approval request and has the following structure:

`name`	*String* Optional. 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.
`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 (e.g., "Pre-State Change").
`reasonForRequest`	*String* Optional. 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:

Value	Description
In Progress	The approval request is pending (i.e., awaiting approval).
Approved	The approval request has been approved.
Rejected	The approval request has been rejected.

Parameter Name	Description		
		No Action	The approval request was auto-approved.
	`submittedDate`	*DateTime* The date on which the request was submitted for approval.	
`scroll`	*Scroll* Optional. 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.

For additional information about working with this structure, see "Getting Details about the Actions of the Approvers Associated with a Request" on page 288.

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

Parameter Name and Description	
`approver`	*String* The key that identifies the user who is the approver.
`statuscomment`	*String* Optional. 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. Can also indicate that the request was auto-approved.