CentraSite

Object Type Management

Version 9.6

April 2014

This document applies to CentraSite Version 9.6.

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

Document ID: IINM-AG-ASSETS-96-20140318

Table of Contents

Preface

This document describes how to use CentraSite Control to create and manage custom types.

The content is organized under the following sections:

What is a Type?	Describes the characteristics and purpose of a type.
What is a Virtual Type?	Describes the characteristics and purpose of a virtual type.
Who Can Create and Manage Types?	Describes the permissions needed to create, edit or delete a type.
Creating a New Type	Describes how to add a custom type to CentraSite.
Viewing or Editing a Type	Describes how to view or modify the definition of a type.
Viewing or Editing a Virtual Type	Describes how to view or modify the definition of a virtual type.
The Predefined Asset Types Installed with CentraSite	Identifies the predefined types installed with CentraSite and describes the kinds of modifications you can make to them.
Customizing the User and Organization Types	Describes the modifications you can make to the User and Organization types installed with CentraSite.
Deleting a Type	Describes how to delete a type.
Working with Composite Types	Explains what a composite type is and identifies the components of the composite types that are installed with CentraSite.
Working with Association Types	Describes how to define and manage association types.

1 What is a Type?

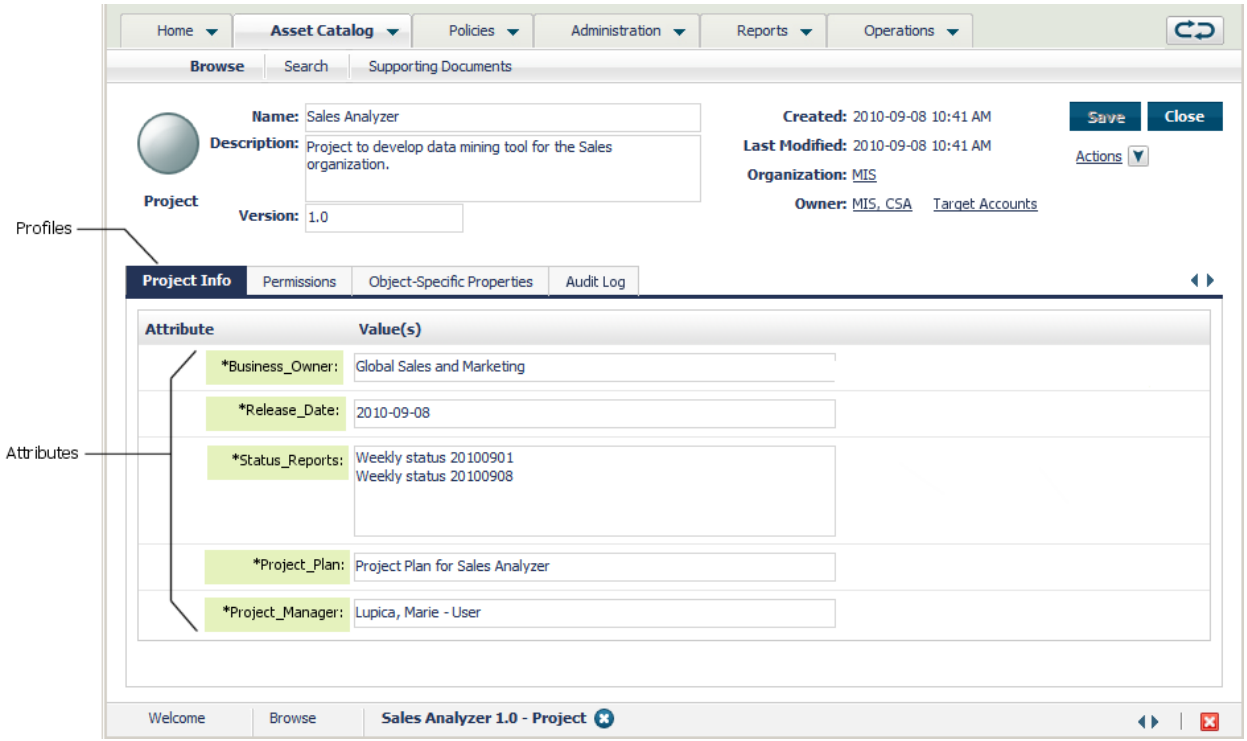
A *type* (also called an object type) is analogous to a class in object-oriented programming and describes a kind of object that the registry can store. Objects abstract the real-world entities and each object belongs to a particular type which defines its characteristics and behavior.

CentraSite includes many predefined types. Any custom type that you add to CentraSite is treated as an *asset type* (i.e., instances of that type are treated as *assets*).

Types are system-wide objects, meaning that they apply to all organizations. Consequently, all organizations within a particular instance of CentraSite use (or have access to) the same global set of types.

A type is made up of *attributes*. Attributes hold data about an object. When a type defines an asset (i.e., if it is an *asset type*), the attributes that make up the type are assigned to *profiles*. Profiles determine how the type's attributes are grouped and presented when an instance of that type is displayed in CentraSite Control or the CentraSite plug-in for Eclipse.

Attributes are grouped by profile when displayed in a CentraSite graphical user interface



Asset types also include an additional set of properties called *advanced settings*. These properties determine how instances of that type are to be managed by CentraSite. Among other things, the advanced settings for an asset type determine whether assets of the type are visible in the catalog browser, whether they can be used with reports and/or policies, and whether they can be versioned.

Types are described in detail in the following sections.

Attributes

An attribute represents an individual characteristic, property or piece of information about an asset. For example, the asset type for a Service includes attributes that identify the name of the service, provide the service's endpoints, identify the owner of the service, supply links to programming documentation and so forth.

All types that represent assets include the following basic attributes:

Attribute	Description
Name	The name under which the asset is cataloged.
Description	A descriptive comment that provides additional information about an asset.
Key	The Universally Unique Identifier (UUID) that is assigned to the asset and uniquely identifies it within the registry. CentraSite automatically assigns a UUID to an asset when the asset is added to the registry.
Version	The user-assigned version identifier for an asset. The user-assigned identifier can be made up of any sequence of characters. It is not required to be numeric.
System Version	The system-assigned version number that CentraSite maintains for its own internal use. CentraSite automatically assigns this identifier to an object when a version of the object is created. The system-assigned identifier is always numeric and always has the format: *MajorVersion*.*Revision* Where: ■ *MajorVersion* is an integer that represents the asset's version number. This value is incremented by one when a new version of the asset is generated (e.g., 1.0, 2.0, 3.0). ■ *Revision* is an integer that represents an update to a particular version of an asset. When the revisioning feature is enabled for CentraSite, the *Revision* number is incremented each time a change is made to the object (e.g., 1.0, 1.1, 1.2). An asset's **System Version** attribute cannot be deleted or modified by a user.
Created	The date on which the asset was added to the catalog. CentraSite automatically sets this attribute when a user adds the asset to the catalog. Once it is set, it cannot be modified.
Last Modified	The date on which the catalog entry for the asset was last updated. CentraSite automatically updates this attribute when a user modifies any of the asset's attributes.
Submitting Organization	The organization to which the asset belongs.
Owner	The user who currently owns the asset. CentraSite automatically sets this attribute when a user adds the asset to the catalog.

Attribute	Description
Lifecycle State	The asset's current lifecycle state. If a lifecycle model has been associated with an asset type, CentraSite updates this attribute as the asset passes through its lifecycle.

An asset can also have any number of additional attributes that are specific to the asset's type. For example, an asset might include attributes that do the following:

- Provide contact information for technical support (for example, phone numbers and email addresses)
- Classify the asset according to one or more taxonomies
- Describe an asset's relationship to other assets or registry objects
- Specify details regarding system requirements and technical specifications
- Provide links to program documentation, sample code, usage notes and so forth

Attribute Data Types

When you add an attribute to a type, you specify the attribute's *data type*. The data type determines what kind of information the attribute can hold. After you add an attribute to a type, the attribute's data type cannot be changed.

The following table lists the data types that you can assign to an attribute. Most types can be configured to hold a single value or multiple values (i.e., an array of values).

Data Type	Description
Boolean	Holds a "true" or "false" value. **Note:** When a Boolean value is displayed in theCentraSite user interface, it's value is generally displayed as "Yes" (if the attribute's value is true) or "No" (if the attribute's value is false) .
Classification	Holds references to one or more categories in a specified taxonomy. You use this type of attribute to classify assets according to a specified taxonomy.
Computed Attribute	Holds a value that is supplied by a user-defined Java plug-in. Once you have defined a computed attribute, you can use it in CentraSite Control in the same way as any other attribute. You can, for example, assign the attribute to a profile or reorder the attribute position within a profile.
Date/Time	Holds a timestamp that represents a specific date and/or time.
Duration	Holds a value that represents a period of time as expressed in Years, Months, Days, Hours, Minutes and Seconds.
Email	Holds an email address. This data type only accepts values in the format: *anyString@anyString*

Data Type	Description
	Note: When a user enters a value for an Email attribute, CentraSite verifies that the value conforms to the format above, but it does not attempt to validate the address itself.
File	Holds references to one or more documents that reside in CentraSite's supporting document library or at a specified URL. You can use this type of attribute to attach documents such as programming guides, sample code and other types of files to an asset.
IP Address	Holds a numeric IP address in the v4 or v6 format.
Multiline String	Holds a string of text. When this type of string is displayed in a CentraSite user interface, the string is displayed in a multi-line text box and lines of text are wrapped to fit the width of the box. (Compare this with the String data type described below.) The **Internationalized** option allows you to store the text in internationalized string format. For more information about the **Internationalized** option, see the String data type, below.
Number	Holds a numeric value. When you define an attribute of this type, you can specify the number of decimal positions that are to be shown when the attribute is displayed in a user interface. If you do not want to restrict the number of decimal positions that the user interface displays, choose the **Maximum Precision** option to display all positions. You can optionally assign a label such as "Seconds", "tps", "KB", "EUR" or "$" to attributes of this type, and specify whether this label is to appear as a prefix or a suffix when the attribute's value is displayed in a user interface. **Note:** The underlying data type for this kind of attribute is a Java double.
String	Holds a string of text. When this type of string is displayed, it is displayed in a single-line text box. If a value exceeds the width of the box, the excess characters are simply not displayed. The **Internationalized** option allows you to create a String attribute that holds different values for different locales. In CentraSite Control, for example, if a user logs on to CentraSite in an English locale and he or she assigns a value to an Internationalized String attribute, that value will be visible to other users with English locales. If a user in a German locale were to view the attribute, the attribute would appear empty because it has no value for the German locale. If the German-locale user were to subsequently assign a value to the attribute, the attribute would then have two String values: one in English and one in German. When CentraSite Control displays an Internationalized String, it displays the value associated with the user's current locale. In the example described above, it would show the English value to users with English locales and the German value to users in German locales. Users in other locales would see an empty attribute until a value for their locale had been assigned to the attribute. The **Enumeration** option allows you to specify a list of allowed values for the attribute.
URL/URI	Holds a URL/URI. This type of attribute only accepts values in the form: *protocol*://*host*/*path*

Data Type	Description
	Where: ■ *protocol* is any protocol that java.net.URL supports ■ *host* is the name or IP address of a host machine ■ *path* (optional) is the path to the requested resource on the specified host
Relationship	Holds references to other registry objects. You use this type of attribute to express a relationship between an asset and another object in the registry.

Attribute Names

An attribute that is one of the following types has two names associated with it: a *display name* and a *schema name*.

Boolean
Date
Duration
Email
Multiline String
IP Address
Number
String
URL

The display name for these types of attributes is the name that is displayed by the CentraSite Control and CentraSite plug-in for Eclipse user interfaces. An attribute's display name can consist of any combination of characters, including spaces.

The following are all valid display names:

```
Business Owner
Amount (in $)
Numéro de téléphone
Avg. Invocations/Minute
1099 Code
```

You can change an attribute's display name at any time.

The attribute's schema name is the name that CentraSite actually gives to the underlying JAXR-based slot that represents the attribute in the registry. This name must be NCName-conformant, meaning that:

- The name must begin with a letter or the underscore character (_).

- The remainder of the name can contain any combination of letters, digits, or the following characters: . - _ (i.e., period, dash, or underscore). It can also contain combining characters and extender characters (e.g., diacriticals).
- The name cannot contain any spaces.

For more information about the NCName type, see **http://www.w3.org/TR/xmlschema-2/#NCName**.

If you do not specify a schema name for an attribute, CentraSite automatically generates a default schema name based on the attribute's display name. It does this by taking the attribute's display name and replacing any spaces in the name with underscore characters (_) and/or by removing any invalid character in the name.

If you explicitly specify a schema name which is not NCName-conformant, CentraSite will request that you change it to an NCName-conformant name.

The following table describes the default schema names that CentraSite would generate for the display names shown above.

For this Display Name…	CentraSite would generate this schema name…	The resulting schema name is…
Business Owner	Business_Owner	*Valid.* You would not need to change the schema name.
Amount (in $)	Amount_in_	*Valid.* You would not need to change the schema name.
Numéro de téléphone	Numéro_de_téléphone	*Valid.* You would not need to change the schema name.
Avg. Invocations Minute	Avg._Invocations_Minute	*Valid.* You would not need to change the schema name.
1099 Code	_099_Code	*Valid.* You would not need to change the schema name.

Note also that an attribute's schema name must be unique within the type (i.e., two attributes in a type cannot have the same schema name).

After the attribute is created, you can no longer change its schema name.

Names for the Other Attribute Types

The concept of display names and schema names does not apply to the following types of attributes:

Classification
File
Relationship

These types are not represented using JAXR-based slots and, therefore, do not require underlying schema names. These attributes have one name, which can consist of any combination of letters, numbers or special characters (including spaces).

Computed Attributes

CentraSite Control offers you the possibility to add computed attributes into asset type definitions and profiles; this allows you to define attributes which require complex computation in Java and then implement them as a Java plug-in, thus overcoming the limitations of predefined attribute types. You could, for example, make attribute values localizable by using computed attributes.

A computed attribute must describe its scale for the rendering within the profile of the asset type.

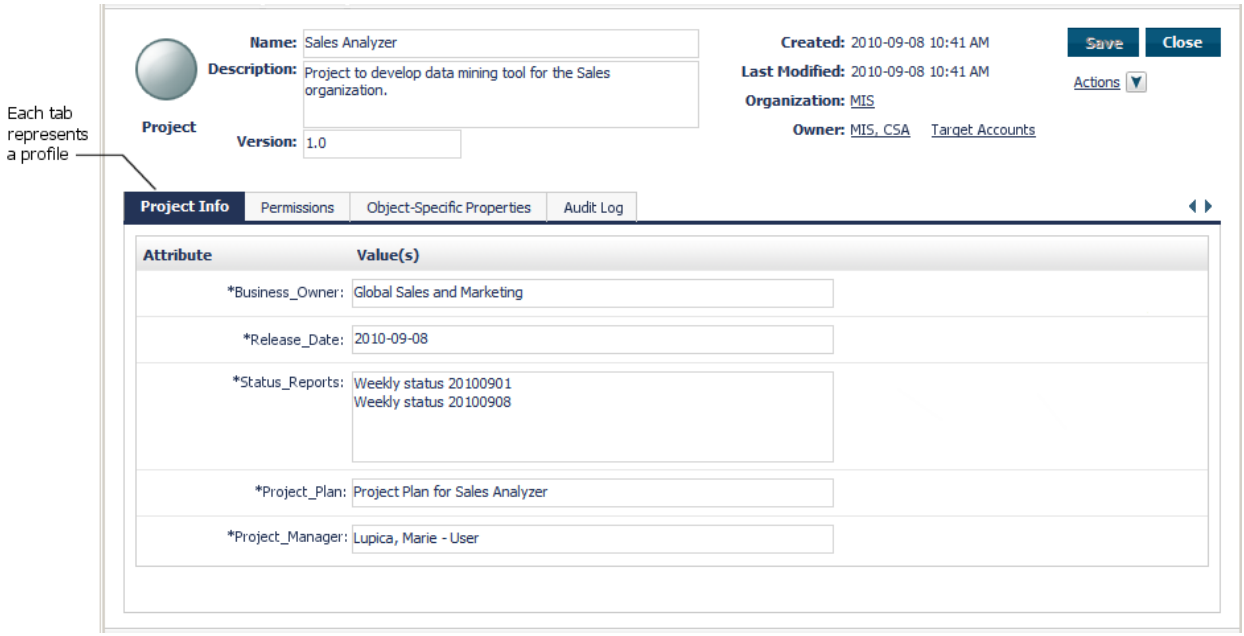
For a Java-based plug-in for a computed attribute, you create a jar file that contains the plug-in definition, and you load the jar file via CentraSite Control into the repository.

After you have added a computed attribute into a profile definition, you can perform administration tasks on the computed attribute in the same way as for normal attributes. For example, you can define the ordering of the attributes in a profile, regardless of whether they are standard attributes or computed attributes.

Profiles

Profiles are used to manage the presentation of attributes in the user interface. They determine how the attributes are grouped when an instance of the asset type is displayed. When you display an asset in CentraSite Control, for example, the attributes associated with a particular profile are grouped together on a tab.

In CentraSite Control, profiles appear as tabs



When you define an asset type, you specify the profiles on which its attributes are to be displayed. CentraSite does not require an attribute to be assigned to a profile. If you do not assign an attribute to a profile, the attribute will not be visible in the user interface (the attribute will still be available via the API). You can assign an attribute to multiple profiles if you want it to appear on multiple profiles (tabs) in the user interface.

You can define any number of profiles for an asset type. You can specify the order in which you want the profiles to appear when an instance of the type is displayed. You can also specify the order in which attributes are to be displayed within each profile.

- Generic Profiles
- Computed Profiles
- Assigning Permissions on Profiles

Generic Profiles

In addition to the profiles that you define, CentraSite provides several predefined profiles, called *generic profiles*, which you can optionally include in an asset type.

The generic profiles contain system-defined attributes and controls. The information on the generic profiles is generated by CentraSite. You cannot customize the content of the generic profiles or add attributes to them. You can, however, select which of these profiles you want CentraSite to include with an asset type.

Generic Profile	Description
Summary	*Available only for Service, XML Schema, Application Server, BPEL Process, Interface and Operation asset types.* Provides basic information about the asset. For a service asset, this profile includes a list of the operations and bindings that the service provides. For a BPEL Process, Interface or Operation asset, the profile shows basic information such as the asset owner, but includes no type-specific attributes.
Consumers	Displays the list of users and applications that are registered to consume the asset. This profile also contains controls for registering an application, a user or a group as a consumer of the asset.
Subscriptions	Displays the list of users who are registered to receive notifications when changes are made to the asset.
Classification	Displays the list of categories by which an asset is classified. This profile also contains controls for adding "ad hoc" classifiers to an asset.
Associations	Displays the list of objects to which the asset is related. This profile also contains controls for establishing "ad hoc" relationships between an asset and other objects in the registry.
Permissions	Displays the asset's instance-level permissions. This profile also contains controls for modifying the asset's instance-level permissions. To view all of the instance-level permissions for an asset, a user must have Modify or Full permissions on the asset. To edit the instance-level permissions for an asset, a user must have Full permissions on the asset. If a user has only View permission on an asset, the Permissions profile shows only that particular user's permissions for the asset.
Versions	Displays the versioning history for an asset and provides links to earlier versions and revisions of the asset. This profile also contains the controls for generating a new version of the asset, purging older versions of the asset and reverting to a previous version of an asset.
External Links	Displays the list of links to external documents and files that are attached to the asset. This profile also contains controls for attaching documents and files to an asset.
Object-Specific Properties	Displays the list of object-specific properties that have been assigned to an asset. An object-specific property consists of a key, which identifies the name of the property, and an optional String value, which contains the data associated with the property. (A property's value can be null.) Object-specific properties are used to hold information about an instance of an asset when there is no predefined attribute to hold that data. Typically, they are used in one-off situations to attach ad-hoc data to an instance of an asset. For example, if you were managing a certification effort, you might use an object-specific property to identify the set of assets that required certification.
Audit Log	Displays the update activity associated with an asset. This profile lists each change that has been made to an asset (including changes in an asset's lifecycle state) and identifies the user that made the change.
Members	*Available for only Package and IS Package types.* Displays the list of objects to which the asset is related. This profile also contains controls for establishing "ad hoc" relationships between an asset and other objects (assets, users, organizations etc) in the registry.

Generic Profile	Description
Performance	Displays the run-time performance metrics associated with an asset. If you are using webMethods Mediator, webMethods Insight or another run-time monitoring component to log performance metrics for an asset, CentraSite displays those metrics on this profile. **Note:** The **Performance** profile is displayed for virtual services regardless of whether you enable or disable it for the Service asset type. In other words, when you disable the **Performance** profile for the Service asset type, CentraSite removes the profile from service assets, but continues to display it for virtual services.
Events	Displays the run-time events associated with an asset. If you are using webMethods Mediator, webMethods Insight or another run-time monitoring component to log run-time events for an asset, CentraSite displays those events on this profile. **Note:** The **Events** profile is displayed for virtual services regardless of whether you enable or disable it for the Service asset type. In other words, when you disable the **Events** profile for the Service asset type, CentraSite removes the profile from service assets, but continues to display it for virtual services.
Policies	Displays the list of design/change-time policies and run-time policies that are applicable to an asset (i.e., it displays all the policies whose scope encompasses the displayed asset). **Note:** The **Policies** profile is displayed for virtual services regardless of whether you enable or disable it for the Service asset type. In other words, when you disable the **Policies** profile for the Service asset type, CentraSite removes the profile from service assets, but continues to display it for virtual services.
Processing Steps	*Available for Virtual Service, Virtual XML Service and Virtual REST Service types.* For a Virtual Service, displays the protocol (HTTP , HTTPS or JMS) and SOAP format (1.1 or 1.2) of the requests that the service will accept. For a Virtual REST Service or Virtual XML Service, displays the protocol (HTTP or HTTPS) of the requests that the service will accept. Also, you specify the HTTP methods (GET, POST, PUT, DELETE or Use Context Variable) that are supported by the native service. This profile also contains the request routing methods and protocol for authenticating the requests.
Deployment	*Available for Virtual Service, Virtual XML Service and Virtual REST Service types.* Displays the list of virtualized services that are ready for deploying in the webMethods Mediator target.

Computed Profiles

CentraSite Control offers you the possibility to add computed profiles into asset type definitions; this gives you the option to define your own profile, which means that you can implement your own algorithms for calculating the values you wish to represent. You could for example aggregate or compute attribute information from embedded or linked objects.

You can combine the attribute specific profile and the generic profiles layout concept in a single computed profile.

Computed profiles let you create your own layout by using a UI Rendering Concept. You can also specify your own rendering logic to display the computed values. You could, for example, create a custom display of performance metrics as a graphic or an animation.

A computed profile can be implemented as a Java plug-in. For a Java-based plug-in for a computed profile, you create an archive file that contains the plug-in definition, and you load the archive file via CentraSite Control into the repository.

After you have added a computed profile into the asset type definition, you can perform administration tasks on the computed profile in the same way as for normal profiles. For example, you can define profile-based permissions, and you can define the order of the computed profile relative to the other profiles in the asset detail display.

Assigning Permissions on Profiles

You can restrict access to individual profiles by setting profile permissions on an instance of an asset. Doing this enables you to control who can view and/or edit the attribute values on a particular instance of a profile. For more information about controlling access to individual profiles within an asset, see *Setting Permissions on an Asset* in the document *Using the Asset Catalog*.

Important: Profile permissions restrict access at the UI level but not the API level. At the API level, profile permissions are irrelevant. If a user has view permission on an asset, he or she can access all of the asset's metadata through the API, regardless of whether profile permissions exist for the asset.

2 What is a Virtual Type?

Certain predefined types installed with CentraSite are classified as *virtual types*. A virtual type is a variant of a specified object type, which is referred to as its *base type*. A virtual type has the same set of attributes as its base type, but has its own set of profiles and properties and adds its own behavior. A virtual type can also have its own lifecycle model and policies.

Virtual types do not have a separate storage structure or a schema, so the instances of a virtual type are stored as regular objects. For example, a Virtual Service inherits the service and adds its own behavior, yet it is stored as Service Object.

Virtualized services are used for entities that are very similar, but generally need to be governed separately. For example, the REST Service type and the XML Service type are both virtual types of the Service type. As services, they have basically the same set of metadata as a Service type, however, you might want to apply different policies to these types of services or manage them using different lifecycle models than a regular Web Service asset.

Several predefined virtual types are installed with CentraSite. For a list of the virtual types that CentraSite provides, see ***The Predefined Asset Types Installed with CentraSite***.

Note: Support for virtual types is limited to those predefined virtual types that are installed with CentraSite. You cannot create custom virtual types of your own.

Virtual types are described in detail in the following sections.

The Properties of a Virtual Type

Virtual types have properties that differ from regular types. The following list describes the ways in which a virtual type differs from a regular object type.

A virtual type inherits all of its attributes from its base type. Therefore, you cannot add attributes directly to a virtual type. To add new attributes to a virtual type, you add the attributes to the base type. You can selectively display these attributes on the profiles that you have defined in the type. Similarly, you cannot delete attributes from or edit the properties of attributes in the virtual type. All attribute creation, deletion, and definition is performed on the base type, and those changes are applied to all of its virtual types.

A virtual type has its own set of Advanced Settings, which enables you to configure the following properties specifically for a virtual type:

- Large and small icons
- Visible in Asset Browse
- Enable Reports
- Policies can be applied
- Require Consumer Registration

- Enable Versioning
- Top Level Type
- Enable Lifecycle Management
- Visible in Search
- Inherit Base Type Profiles
- Inherit Base Type Policies
- Inherit Base Type LCM

A virtual type has an **Inherit Base Type Policies** option, which determines whether the policies of the base type also apply to the virtual type. You can enable or disable this option for each virtual type. For more information about using the **Inherit Base Type Policies** option, see the document *Working with Design/Change-Time Policies*.

A virtual type has an **Inherit Base Type LCM** option, which determines whether the virtual type follows the same lifecycle model as its base type by default. You can enable or disable this option for each virtual type. For more information about using the **Inherit Base Type LCM** option, see *Customizing Lifecycle Management*.

A virtual type can have its own set of profiles:

- A virtual type has an **Inherit Base Type Profiles** option, which determines whether the profiles of the base type also apply to the virtual type. You can enable or disable this option for each virtual type. For more information about using the **Inherit Base Type Profiles** option, see ***Applying Profiles to Virtual Types***

Like many other predefined types, you cannot delete the virtual predefined types that are installed with CentraSite.

Using the Inherit Base Type Profiles, LCM and Policies Options

The **Inherit** options for a virtual type determines whether the profiles, lifecycle models and policies associated with the base type are also applied to the virtual type.

For more information about how each of these options affect a virtual type, see the following sections:

- ***Applying Profiles to Virtual Types***
- *Applying Lifecycle Models to Virtual Types* in the document *Customizing Lifecycle Management*
- *Applying Policies to Virtual Types* in the section *Functional Scope* in the document *Working with Design/Change-Time Policies*

By default, the **Inherit** options are enabled for each of the predefined virtual types installed with CentraSite.

3 Who Can Create and Manage Types?

To create custom types, you must belong to a role that has the "Manage Asset Types" permission. Besides allowing you to create custom types, this permission allows you to edit and delete any user-defined asset type. Additionally, it allows you to edit certain predefined types that are installed with CentraSite. By default, users in the CentraSite Administrator or Asset Type Administrator role have this permission, although an administrator can grant this permission to other roles.

For more information about permissions, see *About Roles and Permissions* in the document *Users, Groups, Roles and Permissions*.

4 Creating a New Type

CentraSite provides a wizard for defining new types. When creating a new type, keep the following points in mind:

- A type has two names: a *display name* and a *schema name*.
  - The display name is the name that CentraSite uses when it refers to the type in the user interface. (This is the name that appears in the catalog browser, for example.) The display name can contain any character, including spaces.
  - The schema name is the name that is given to the underlying schema that contains the type definition. The schema name must conform to the naming requirements for an NCName data type, which does not permit names with spaces or most special characters.

  By default, CentraSite derives the schema name from the display name that you specify for the type. If your display name includes special characters, then you must explicitly specify a schema name that is NCName conformant in the type's **Advanced Settings** dialog box.
- If you intend to include a Classification attribute in your custom type, make sure that the corresponding taxonomy exists before you begin creating the new type. To add a Classification attribute to a type, you must specify the taxonomy with which the attribute is associated. You cannot do this unless the taxonomy already exists in CentraSite. For information about creating taxonomies, see the document *Taxonomies*.
- If you intend to include a Relationship attribute in your custom type, make sure that the corresponding association type exists before you begin creating the new type. To add a Relationship attribute to a type, you must specify the type of association that the attribute represents. You cannot do this unless the association type is already defined CentraSite. For information about defining association types, see ***Working with Association Types***.

The following sections describe how to create a new type.

Creating the New Type

Use the following instructions to create a new type:

To create a new type

1 In CentraSite Control, go to **Administration > Types**.

2 On the **Types** tab, click **Add Asset Type** to open the **Add Asset Type** wizard.

3 In panel 1, complete the following fields:

In this field...	Do the following...
Name	Enter a display name for the type. Choose a name that your users will recognize and understand. For example, use "BPEL Process Document", not "bpdoc". The name you assign to the asset type can contain any character, including spaces. However, if you specify a name that does not conform to the NCName type, you *must* click the **Advanced Settings** button and specify a name that is NCName conformant in the **Schema Name** field. For NCName requirements, see the **Schema Name** field description in the next step. **Note:** If the name that you assign to the asset type is NCName conformant, except that is includes spaces, it is not necessary to explicitly specify the type's schema name. CentraSite automatically replaces space characters with the _ character when it generates the schema name for an asset type.
Description	*Optional*. Enter a brief description of the type.

4 Click **Advanced Settings** and complete the following steps as necessary.

1. Verify that the schema name and the namespace name that were generated by CentraSite are valid.

In this field...	Do the following...
Schema Name	Verify that the schema name that CentraSite generated for this asset type is NCName conformant. CentraSite automatically populates this field with a name that is derived from the asset type name that you specified in the previous step. For example, if your asset type name is "My Asset Name", CentraSite automatically populates this field with "My_Asset_Name". If the schema name generated by CentraSite does not meet the following criteria, you must specify a name that does. ■ The name must begin with a letter or the underscore character (_). ■ The remainder of the name can contain any combination of letters, digits, or the following characters: . - _ (i.e., period, dash, or underscore). It can also contain combining characters and extender characters (e.g., diacriticals). ■ The name cannot contain any spaces. Additionally, the type's fully qualified schema name must be unique among all types in the registry. See **Namespace**, below. For more information about the NCName type, see http://www.w3.org/TR/xmlschema-2/#NCName. **Note:** You cannot change the schema name after the type is created.
Namespace	Modify the namespace that CentraSite has proposed for the type if necessary. By default, CentraSite generates the namespace in the following form:

In this field...	Do the following...
	`http://namespaces.`*`OrganizationName`*`.com/Schema` Where, *`OrganizationName`* is the name of your organization. The **Namespace** value is used to qualify the name specified in **Schema Name**. Together, the **Schema Name** and **Namespace** values produce the type's fully qualified name. This name must be unique within the registry. You generally do not need to modify the namespace that CentraSite proposes for a type. In most cases, the proposed namespace will be adequate. However, you might modify the namespace if you want it to include the name of a different organization or if you need to resolve a naming conflict between this type and an existing type. **Note:** You cannot change the namespace value after the type is created.

2. If you want to use a custom icon to represent this type in the user interface, upload large and small versions of the icon as described below.

   Note: If you do not specify a custom icon, CentraSite assigns the default icon to the type.

In this field...	Do the following...
Large Icon	*Optional*. Specify the large icon that is to be used to represent this type. CentraSite Control uses this icon when it displays the details for an instance of the type. If you want to use a custom icon, click the **Browse** button and upload the file containing the large version of the icon to CentraSite. The icon must be in GIF format. To ensure proper alignment when it is displayed in the user interface, the icon must be 64 x 64 pixels in size.
Small Icon	*Optional*. Specify the small icon that is to be used to represent this type. CentraSite Control displays this icon when instances of the type appear in lists or summary tables. If you want to use a custom icon, click the **Browse** button and upload the file containing the small version of the icon to CentraSite. The icon must be in GIF format. To ensure proper alignment when it is displayed in the user interface, the icon must be 16 x 16 pixels in size.

3. Specify the type's advanced options as follows:

Enable this option...	To...
Visible in Asset Browse	Allow instances of this type to be displayed in the catalog browser. When you enable this option, CentraSite Control includes the type in the **Asset Types** pane on the **Asset Catalog > Browse** page. When you disable this option, CentraSite Control omits the type from the **Asset Types** pane so users cannot browse for instances of the type.
Enable reports	Allow reports to be generated against instances of this type.
Policies can be applied	Allow user-defined design/change-time policies to be created and enforced for instances of this type. **Note:** If you disable this option, CentraSite will not apply any user-defined design/change-time polices to instances of the type, even in cases where the policy is designed to execute against *all* asset types. **Note:** This option does not apply to system policies. CentraSite will apply system policies to instances of the type whether this option is enabled or not.
Require Consumer Registration	Require users to register an application when they submit consumer registration requests for assets of this type.
Enable versioning	Allow users to generate versions of instances of this type. When you disable this option, CentraSite disables the **Add New Versions** command and omits the **Versions** profile from instances of this type.
Top Level type	Allow users to create instances of this asset type "from scratch." When you enable this option, users are allowed to create instances of the type using the **Add Asset** button in CentraSite Control. Generally you disable this option for types that are constituents of other assets, or for types that are only meant to be added to the registry by an importer. For example, the Operation type is used to represent an operation that belongs to a Web service. Operations are derived automatically from the service WSDL. They are not intended to be manually defined by users. Therefore, the Operation type is not designated as a "top level type".
Enable Lifecycle Management	Allow a lifecycle model to be applied to assets of this type.
Visible in Search	Allow users to define a search for this particular type. When you enable this option, CentraSite Control includes the type in the **Types** list on the Advanced Search page. Including a type in this list enables users to define queries that select on that specific type. **Note:** If you change the state of this option and then do an advanced search, you might need to refresh the Advanced Search page in order to see the change reflected in the **Types** list.
Inherit Base Type Profiles	*For virtual types only.* Apply the set of profiles that are defined for the virtual object type and/or the set of profiles that are defined for the base type.
Inherit Base Type Policies	*For virtual types only.* Apply the set of policies that are defined for the virtual object type and/or the set of policies that are defined for the base type.

Enable this option...	To...
Inherit Base Type LCM	*For virtual types only.* Apply the lifecycle model defined for the base type if the virtual object type does not have an assigned lifecycle model.

4. Click **OK**.

5 In panel 2, specify the attributes that make up the type. If you need procedures for this step, see *Defining Attributes for a Type*.

6 In panel 3, define profiles for the type and assign attributes to them. If you need procedures for this step, see *Defining Profiles for an Asset Type*.

7 In panel 4, specify which of the generic profiles you want to include with the type. For more information about generic profiles, see *Generic Profiles*.

8 In panel 5, specify the order in which you want the profiles to appear when they are displayed in the user interface.

9 Click **Finish** to save the new type.

Defining Attributes for a Type

General Procedure

Use the following steps to complete panel 2 in the **Add Asset Type** or **Edit Asset Type** wizard. You use this panel to define the set of attributes that make up the asset type.

Using the Dialog

▶ **To define attributes for an asset type**

1 Go to panel 2 in the **Add Asset Type** wizard (if you are creating a new type) or the **Edit Asset Type** wizard (if you are modifying an existing type). If you need procedures for this step, see *Creating a New Type* or *Viewing or Editing a Type*, respectively.

2 To add a new attribute to the type, click **Add Attribute**.

Complete the following fields in the **Add Attribute** dialog box:

In this field...	Specify...
Name	The display name for the attribute. This is the attribute name that users will see when they view instances of this type in the user interface, therefore, the name should be meaningful. The display name can contain any combination of characters, including spaces. You can change an attribute's display name at any time. **Note:** If you are defining a Relationship attribute, by default the attribute's name will be derived from the name of the association type that you assign to the attribute. You can, however, assign a custom name to the Relationship attribute by specifying the **Name** attribute.
Schema Name	The schema name for the attribute. The schema name is the internal name that CentraSite assigns to the attribute. This name can contain only letters, numbers and the underscore character (_). If you do not enter a schema name, CentraSite automatically generates a schema name for the attribute based on the name you enter in the **Name** field. However, if the name that CentraSite generates includes invalid characters, you will be prompted to provide a valid schema name when you save the attribute. For information about valid schema names, see *Attribute Names*. After you create and save the attribute, you can no longer change its schema name. **Note:** Relationship, File, and Classification attributes do not require schema names.
Description	*Optional.* A brief description for the attribute.
Required	Whether the attribute is required or optional. When you enable this option, CentraSite prevents users from saving an asset of this type without first assigning a value to this attribute. In the user interface, a required attribute is displayed with an asterisk (*) next to its name. **Note:** An attribute can be a required attribute and have a default value (see the **Default** field below). If you do not supply a value for an attribute that is required and has a default value, the default value is automatically assigned to this attribute. When editing an existing type: ■ If there are no instances of the type in the registry, you can add a required attribute to an existing type. ■ If there are instances of the type in the registry, you can add a required attribute of type "slot" or "classification", but not of type "relationship" or "file". ■ If you add a new required attribute, no automatic update of existing instances takes place. This prevents the potential degradation of performance that could arise from the automatic update of a large number of instances. ■ You can enable the **Required** option for an existing attribute even if there are empty instances of the attribute. But in this case a default value must be provided for the attribute.

In this field...	Specify...
	■ You can disable the **Required** option for an attribute at any time (even if assigned instances of the attribute already exist in the registry). See also the description of the **Default** field below for details of default value processing of required fields.
Read-only	Whether users can modify the value of the attribute. The **Read-only** option prevents the attribute's value from being changed after an asset is created. When you enable this option, users can assign a value to the attribute when an asset of this type is initially created. However, they cannot modify the attribute after the asset has been saved. You can enable or disable the **Read-only** option for an attribute at any time (even if instances of the type already exist in the registry).
Multiplicity	Whether the attribute can hold a just a single value or multiple values (i.e., an array of values). Enabling the **Multi Value** option allows users to assign more than one value to the attribute. **Note:** The **Multiplicity** option is not available for the Boolean attribute type. When editing an existing asset type: ■ You can switch an attribute's **Multiplicity** option from **Single Value** to **Multi Value** at any time (even if instances of the type already exist in the registry). ■ You can switch an attribute's **Multiplicity** option from **Multi Value** to **Single Value** only if: ■ No instances of the type exist in the registry. —OR— ■ Instances of the type exist in the registry, but each instance has at most one value assigned to this attribute. (i.e., no instances exist wherein this attribute has multiple values).
Data Type	The attribute's type. Be aware that after you create an attribute, you cannot change its type. Depending on the data type you select, additional type-specific fields or checkboxes now appear in the dialog. Provide the appropriate information for these additional fields or checkboxes. See the section **Defining Data Types** below for details.
Default Value	*Optional*. The value that is to be assigned to the attribute by default. **Note:** The **Default Value** option is not available for all attribute types. When editing an existing asset type:

In this field...	Specify...
	■ You can change a default value, assign a default value, or remove a default value from an attribute at any time. Changing the attribute's default value *does not* immediately affect any existing instances of the attribute. ■ If you add a default value to an attribute that did not previously have one, and the registry contains empty instances of that attribute, the default value will be assigned to those assets the next time that they are saved to the registry. ■ If you add a new attribute to an existing type and you assign a default value to that attribute, the default value will be assigned to the existing instances of that type the next time those instances are saved to the registry. If the attribute has the **Required** option set, then the following conditions apply: ■ When you create a new asset type with one or more required attributes, you do not need to provide a default value for these attributes during the creation process. ■ If instances of an asset type exist, and you update the type definition in any manner, regardless of whether you edit the required attributes or not (for example, adding a new optional attribute, adding a new required attribute, or even editing the type name), you must provide a default value for each required attribute of that type. Required attributes that have no value will be set to the default value the next time the instance is saved to the registry. ■ If instances with missing required attributes are viewed in CentraSite Control, these attributes are simulated and displayed with the default value. But the default value will not be added to the instance until the next save of the instance. See also the description of the **Required** field above for details of default value processing of required fields.

For the **Data Type** field, complete the type-specific options as described in the section **Defining Data Types** below.

Click **OK** to save the attribute.

3 Repeat step 2 for each attribute that you want to add to the asset type.

4 If you need to edit an attribute that you have added to the asset type, click the name of the attribute in panel 2 and make your changes in the **Edit Attribute** dialog box. Be aware that certain options for an attribute cannot be changed after the attribute is created. For information about the conditions under which an attribute's options can be changed, see the descriptions of the options in step 2, above.

5 If you need to delete an attribute, select the attribute in panel 2 and click **Delete**.

Important: CentraSite will not allow you to delete an attribute if an assigned instance of the attribute is present in the registry.

Defining Data Types

For the **Data Type** field, complete the type-specific options as described below:

If you are adding this type of data type...	Do the following...
Classification	In the **Taxonomy** field, specify the taxonomy by which users will classify the asset. **Note:** After you create a classification attribute, neither its **Taxonomy** option or its **Default** option can be changed.
Computed Attribute	When you choose this data type, you are indicating that the attribute value, and the way it is rendered in a profile, will be generated using a Java plug-in. Choose **Java** as the implementation type, to indicate that you are using a Java plug-in. In the field **File**, specify the name of the archive file that contains the Java plug-in. See the section **Defining a Computed Attribute** below for more information. **Note:** When you define a computed attribute, the dialog fields **Required**, **Read-Only** and **Multiplicity** are not relevant, so they are automatically removed from the dialog.
Date/Time	In the **Display As** field, specify whether you want the user to specify: ■ The date and time ■ The date only ■ The time only
Relationship	Using this data type, you specify that an asset of the type you are currently defining can be related to another asset or object. You define the nature of the relationship in the **Relationship Type** field, using one of the following options: ■ Association ■ Aggregation using Source ■ Aggregation using Target ■ Composition using Source ■ Composition using Target These options allow you to define a loose or tight coupling between the related objects. Depending on the type of relationship, operations on an asset (such as delete, export, move to another organization, set instance-level permissions, create a new version) can be applied automatically also to the related asset or object. See the section **Working with Composite Types** for information on how you can use these relationship types.

If you are adding this type of data type...	Do the following...
	In the **Association Type** field, specify the type of association that this attribute represents. This is a label that you assign as a meaningful name for the relationship. Examples of labels are "hasChild" and "hasParent", indicating a hierarchical relationship. In the **Relates To** field, specify the type(s) of object that can be the target of the relationship. You can specify more than one type of object by using the **+** icon. If you specify more than one type of object in this field, this means that an object of the current object type can have a relationship to an object that belongs to any of the given object types. **Note:** The **Relates To** option is enforced in the user interface and at the API level. Within the user interface, this option determines which types of object are shown to users when they set the attribute. At the API level, this option causes CentraSite to reject any asset whose attribute specifies an object that is not of an allowed type. After you create a Relationship attribute, you can modify the attribute's **Association Type** *only* if there are no assigned instances of the attribute in the registry. Similarly, you can change or delete categories in the **Relates To** option *only* if the registry contains no assigned instances of the attribute. You can, however, assign additional categories to the **Relates To** option at any time (even if assigned instances of the attribute exist in the registry). For more information about association types, see ***Working with Association Types***.
String or Multiline String	Enable the **Internationalized** option if you want to store the string values in internationalized string form. For more information about the Internationalized option, see the description of the String attribute in ***Attribute Data Types***. After you create a String or Multiline String attribute, you cannot change the attribute's **Internationalized** option.
String	Enable the **Enumeration** option if you want to restrict the attribute to a predefined set of values. When users edit this attribute in CentraSite Control, the enumerated values are presented to them in a drop-down list. CentraSite also enforces the **Enumeration** option at the API level, meaning that it will reject any asset whose attribute does not contain one of the enumerated values. If you enable the Enumeration option for a String attribute, you can add new items to the enumerated list at any time (even if instances of the attribute exist in the registry). However, you can remove an item from the list only if that item is not currently assigned to an instance of this type in the registry. After you create a String attribute, you cannot switch it from an enumerated String to a non-enumerated String (or vice-versa).

Defining a Computed Attribute

The computed attribute is defined as the implementation of the `ComputedAttribute` interface. The following topics describe how to define a computed attribute:

- Implementation guidelines for Java-based computed attributes
- Structure of archive file
- Sample code
- Loading a computed attribute into an asset type definition

Implementation guidelines for Java-based computed attributes

This section describes the Java interfaces and methods that you need to implement for a computed attribute.

Interfaces	Description
ComputedAttribute	This interface declares basic rendering methods for the user interface. It extends the ProfileAttribute interface. boolean: isUsed(): returns true if this attribute is used by at least one instance of the corresponding asset type. Collection: setValue(Collection): sets the value of this attribute. AttributeDescriptor: getAttributeDescriptor(): returns the definition of this attribute. String: getName(): returns the JAXR-based name of this attribute. Collection: getValue(): returns the value of this attribute. init(): adds the required parameters to the objects argument for attribute initialization. Collection<Concept>: getTargetObjectTypes(): returns the appropriate target registry object of this attribute. This method will be used when the attribute is of the relationship data type. Concept: getAssociationType(): returns the appropriate association type of this attribute. This method will be used when the attribute is of the relationship data type. ClassificationScheme: getTaxonomy(): returns the appropriate classification scheme of this attribute. This method will be used when the attribute is of the classification data type.
ProfileAttribute	void: init(Collection<CentraSiteRegistryObject>, Locale) Collection: getValue(); String: getAttributeKey(); AttributeDescriptor: getAttributeDescriptor();

Interfaces	Description
AttributeDescriptor	This interface deals with the standard properties (isReadOnly(), is Required()...) for an attribute. String: getDataType(): returns the data type of this attribute. This can be one of the supported standard types (xs:...), or one of the special types: File, RichText, Image (allows the specification of the image type). Object: getDefaultValue(): returns the default value of this attribute. String[]: getEnumValues(): int: getMaxLength(): returns the maximum length of this attribute. String: getMaxOccurs(): returns whether the attribute can have multiple occurrences. String: getMinOccurs(): returns whether the attribute is optional or required. Object: getNativeAtribute(): returns the native attribute instance if the data type is not primitive, otherwise returns null. int: getPrecision(): gets the precision for a "number" attribute. String: getUnitLabel(): returns the slot's unit label. boolean: hasDefaultValue(): returns whether this attribute has a default value. boolean: isPrefix(): Returns true if the slot's unit label is a prefix, and false if it is a suffix. boolean: isReadOnly(): returns whether this attribute is read-only. boolean: isRequired(): returns whether this attribute is required.

Structure of archive file

The archive file must contain the following folders and files:

Zip folder	Description
META-INF	This folder contains the *config.properties* file, which is the build file for the plug-in. This properties file contains an entry of the following format:

Zip folder	Description
	`com.softwareag.centrasite.computed.attr.impl.class=com.sample.StringAttrImpl`
lib	This folder contains the archive file with the source code examples, the plug-in's executor class and the external libraries.

Sample code

Sample Java code for implementing a computed attribute is supplied in the *demos* folder under CentraSite installation folder. The sample code is available in the files:

- ComputedAttribute.java
- ProfileAttribute.java
- AttributeDescriptor.java

Loading a computed attribute into an asset type definition

After you have created an archive file that contains the attribute definition, you need to load the archive file into the asset type definition. You do this by starting the **Edit Asset Type** wizard for the appropriate existing asset type, or the **Add Asset Type** wizard for a new custom asset type, and specifying in the wizard that you are defining a new computed attribute.

For details on how to load the archive file of a computed attribute into an asset type definition, refer to the section ***Defining Attributes for a Type***.

When you have loaded the archive file, the new attribute is displayed in the list of attributes that can be assigned to an asset type profile.

Defining Profiles for an Asset Type

- Applying Profiles to Virtual Types

General Procedure

Use the following procedure to complete panel 3 in the **Add Asset Type** or **Edit Asset Type** wizard. You use this panel to define profiles and their attributes.

You can add a profile in two ways:

- **Manually, using the asset type's available attributes.**
  To define a profile manually, you select attributes from the list of available attributes and assign them to the profile.
- **Using a Java plug-in that contains a computed profile.**
  A computed profile is a user-defined profile that is implemented as a Java plug-in. You add the computed profile to the asset type by importing the computed profile's definition from an archive file. The plug-in specifies the attributes that are contained in the profile. The plug-in also has the sole responsibility for rendering the layout representation within the profile. After a computed profile has been defined for an asset type, the computed profile is treated in the same way as any other profile; for example, permissions for computed profiles can be granted in the same way as for standard profiles, and the ordering of profiles within a type definition is the same for computed profiles as for standard profiles.

To add a profile to the asset type

1 Go to panel 3 in the **Add Asset Type** wizard (if you are creating a new type) or the **Edit Asset Type** wizard (if you are modifying an existing type). If you need procedures for this step, see *Creating a New Type* or *Viewing or Editing a Type*, respectively.

2 To add a new profile to the type, click **Add Profile** and perform the following steps.

1. In the **Profile Name** field, enter a name for the profile. This is the name that users will see in the user interface when they view an asset of this type. Therefore, the name you assign should be meaningful to your users (for example, "Technical Notes", not "tn").

   Note: You cannot give a profile the same name as any of the generic profiles described in *Generic Profiles*.

2. If the profile you wish to add is a computed profile, mark the checkbox **Computed Profile** and select the implementation language from the drop-down list. If, for example, you have implemented the computed profile using Java, then select **Java**.

   When you mark the checkbox, the dialog's list of available attributes is replaced by the input field **Profile Implementation Archive**. In this field, specify the name of the archive file that contains the definition of the computed profile, then click **OK**.

See the section **Defining a Computed Profile** below for information on how to implement a computed profile.

Note: If you define a profile as a computed profile, you cannot change the profile back to being a non-computed profile at a later stage. You can, however, change the archive file that contains the definition of the computed profile.

3. If your new profile is not a computed profile, continue with this step.

   Use the arrow buttons to specify the attributes for the new profile and the order in which the attributes will be displayed.

   - Add an attribute to the profile by marking the attribute's checkbox in the list of available attributes and clicking the right-pointing arrow.
   - Add all of the available attributes to the profile by clicking the right-pointing double arrow.
   - Remove an attribute from the profile by marking the attribute's checkbox in the list of assigned attributes and clicking the left-pointing arrow.
   - Remove all attributes from the profile by clicking the left-pointing double arrow.
   - Change the position of an assigned attribute relative to the other assigned attributes within the profile by marking the checkbox of the attribute and clicking the up or down arrow as required.

4. Click **OK**.

3 If you need to edit a profile that already exists within the type, click the name of the profile in panel 3 and make your changes in the **Edit Profile** dialog box.

Use the arrow buttons, as described in the previous step for **Add Profile**, to specify the attributes for the profile, and the order in which the attributes will be displayed.

4 If you need to delete a profile, select the profile in panel 3 and click **Delete**.

Note: Deleting a profile *does not* delete the attributes that were assigned to the profile. It simply removes the profile definition from the asset type. The attributes still exist within the type. However, they will not be visible in the user interface unless you assign them to another profile.

Defining a Computed Profile

The following topics describe how to define a computed profile:

- Definition of Java-based Computed Profile
- Implementation guidelines for Java-based computed profiles
- Structure of archive file
- Sample computed profiles
- Loading a computed profile into an asset type definition

Definition of Java-based Computed Profile

A Java based computed profile has the following rendering mechanism:

- *WithUiRendering*: This dictates the user-defined rendering of the profile's attributes.
- *WithoutUiRendering*: This dictates the CentraSite's default rendering of the profile's attributes. This default rendering is based on the attribute's data type.

Implementation guidelines for Java-based computed profiles

This section describes the Java interfaces and methods that you need to implement for a computed profile.

Interfaces	Description
ComputedProfile	This interface declares basic rendering methods for the user interface. void: init(java.lang.Object, Locale): with `CentraSiteRegistryObject` as a parameter where the necessary implementation is done and updateAsset() which would return a collection of registry object serves as a save hook. boolean: canRenderUI(): This determines whether the rendering is based on the UI (true) or on the triples associated with the profile (false). Collection: getAttributes(): returns a collection of `ProfileAttribute` and would be called only when canRenderUI() returns true. Collection: updateAsset(): returns a collection of `CentraSiteRegistryObject` and would be called only when canRenderUI() returns true.
WebUIProfile	This interface is specific to the rendering for CentraSite Control. java.lang.Object: createProfileContent(): create the profile contents with XML content (in compliance with Application Designer).
EclipseProfile	This interface is specific to the rendering for Eclipse Designer.

Interfaces	Description
	createFormContent(): create the profile contents on the supplied IManagedForm (in compliance with Eclipse Designer SWT/JFace library).
BUIProfile	This interface is specific to the rendering for CentraSite Business UI. getEditPageURL(): returns URL of the edit page of computed profile and would be called only when canRenderUI() returns true. getViewPageURL(): returns URL of the view page of computed profile and would be called only when canRenderUI() returns true. getProfileDataAsJson(): returns a collection of profile data as JSON-formatted string. Collection computeProfileData (String userInputsAsJSON): sets a collection of profile data as JSON-formatted string. JSON specification is available at **http://www.json.org/**.
ProfileAttribute	This interface deals with the standard UI rendering of the profile's attributes and allows defining the attributes as key value pairs. The rendering of each attribute will be the standard rendering for the corresponding datatype of the attribute. AttributeDescriptor: getAttributeDescriptor(): returns the descriptor of the attribute. String: getAttributeKey(): returns the key of the attribute. String: getName(): returns the name of the attribute. Collection: getValue(): returns the value of the attribute.
ComputedAttributeLine	This interface deals with the user-defined UI rendering of the profile's attributes. The UI rendering of the attributes will be determined by the coding of this interface. void: buildUI(StringBuilder layout): This method is responsible for rendering the layout definition. void: passivate(): This method is responsible for storing the values back to the object. void: revert(): This method is to revert the changes.
AttributeDescriptor	This interface deals with the standard properties (isReadOnly(), is Required()...) for an attribute. String: getMinOccurs(): returns the minimum allowed occurrences of this attribute.

Interfaces	Description
	String: getMaxOccurs(): returns the maximum allowed occurrences of this attribute. boolean: isRequired(): returns whether this attribute is required. boolean: isReadOnly(): returns whether this attribute is read-only. boolean: hasDefaultValue(): returns whether this attribute has a default value. Object: getDefaultValue(): returns the default value of this attribute. int: getMaxLength(): returns the maximum length of this attribute. String: getDataType(): returns the data type of this attribute. Object: getNativeAtribute(): returns the native attribute instance if the data type is not primitive, otherwise returns null. String[]: getEnumValues(): boolean: isPrefix(): returns whether this slot's unit label is a prefix or suffix value. Return "true" if the unit label is a prefix, "false" for suffix. int getPrecision(): returns the precision for a "number" attribute. String: getUnitLabel(): returns this slot's unit label. The label may be null.

Structure of archive file

The archive file must contain the following folders and files:

Zip folder	Description
META-INF	This folder contains the *config.properties* file, which is the build file for the plug-in. This properties file conta an entry of the following format: `com.softwareag.centrasite.computed.profile.webui.impl.class=com.sample.MyProfileIm`
lib	This folder contains the archive file with the source code examples, the plug-in's executor class and the exter libraries.

Sample computed profiles

Your CentraSite installation contains 5 sample computed profiles. Two samples are for the CentraSite Control, one for the CentraSite Business UI, and the other two are for the Eclipse Designer.

The content is organized under the following sections:

- Sample Computed Profiles for CentraSite Control
- Sample Computed Profiles for CentraSite Business UI
- Sample Computed Profiles for Eclipse Designer

Sample Computed Profiles for CentraSite Control

Your CentraSite installation contains two sample computed profiles (which is contained in *demos* folder) that you can use to create an archive file for the computed profile specific to CentraSite Control.

- *NonPrimitiveDataTypeSamples*
- *SampleComputedProfile*

Sample Computed Profiles for CentraSite Business UI

Your CentraSite installation contains a sample computed profile (which is contained in *demos* folder) that you can use to create an archive file for the computed profile specific to CentraSite Business UI.

- *SampleProfile*

Sample Computed Profiles for Eclipse Designer

Your CentraSite installation contains two sample computed profiles (which is contained in *demos* folder) that you can use to create an archive file for the computed profile specific to Eclipse Designer.

- *DesignerProfileSample*
- *SampleComputedProfile*

Loading a computed profile into an asset type definition

After you have created an archive file that contains the profile definition, you need to load the archive file into the asset type definition. You do this by starting the **Edit Asset Type** wizard for the appropriate asset type and specifying in the wizard that you are defining a new computed profile.

For details on how to load the archive file of a computed profile into an asset type definition, refer to the section *Defining Profiles for an Asset Type*.

When you have loaded the archive file, the new profile is displayed in the detail page of all assets of the asset type.

Applying Profiles to Virtual Types

When an asset is an instance of a virtual type, the set of profiles that CentraSite applies to the asset depends on the virtual type's **Inherit Base Type Profiles** setting. If the type's **Inherit Base Type Profiles** option is enabled, CentraSite applies the profiles of the base type to the asset *in addition to* the profiles of the virtual type. For example, the **Inherit Base Type Profiles** option is, by default, enabled for virtual services. Therefore, when CentraSite applies profiles for a virtual service, it applies the set of profiles that are defined for the Virtual Service object type *and* the set of profiles that are defined for the Service object type (the base type for the Virtual Service type).

If you disable the **Inherit Base Type Profiles** option for a virtual type, CentraSite applies to the asset only the profiles that are defined for the virtual type. Therefore, CentraSite applies to virtual services only the profiles that are defined for the Virtual Service type. Profiles that are defined for the Service type are not applied.

The following table summarizes how the set of profiles that CentraSite applies for a virtual type is affected by the state of the **Inherit Base Type Profiles** option.

If the virtual type's "Inherit Base Type Profiles" option is...	And the profile is defined for the...	The instances of the virtual type will have...
ENABLED	Base Type	Base Type Profiles
ENABLED	Virtual Type	Virtual Type Profiles
ENABLED	Base Type —AND— Virtual Type	Base Type Profiles —AND— Virtual Type Profiles
DISABLED	Base Type	None
DISABLED	Virtual Type	Virtual Type Profiles
DISABLED	Base Type —AND—	Virtual Type Profiles

If the virtual type's "Inherit Base Type Profiles" option is...	And the profile is defined for the...	The instances of the virtual type will have...
	Virtual Type	

For information about virtual types, see ***What is a Virtual Type?***. For information about which predefined types in CentraSite are virtual types, see ***The Predefined Asset Types Installed with CentraSite***.

5 Viewing or Editing a Type

You use the **Edit Asset Type** wizard to view or edit the properties and options for an existing asset type. When editing an existing type, keep the following points in mind:

- You can modify the type's display name, description, icons and advanced options. You cannot modify the type's **Schema Name** or its **Namespace** property. These two properties are set when the type is created and cannot be changed thereafter.
- With respect to attributes:
  - You can add new optional attributes to the type at any time.
  - You can add new required attributes if there are no existing instances of the type in the registry. If there are existing instances of a type, you can add a new required attribute of type "slot" or "classification", but not of type "relationship" or "file".
  - If you add a new attribute, regardless of whether it is optional or required, no automatic update of existing instances takes place. This prevents the potential degradation of performance that could arise from the automatic update of a large number of instances.
  - You cannot modify the data type for an attribute, but you can modify many of an attribute's other options. Be aware that certain are not permitted if assigned (i.e., non-empty) instances of the attribute are present in the registry. For information about the kinds of changes you can make to an existing attribute, see the property and option descriptions in *Defining Attributes for a Type*.
  - You cannot delete an attribute from a type if an instance of the type exists with a value assigned to the attribute. In such a situation, you must first remove the attribute's value from each such instance before you can delete the attribute.

    Note that the command line tool "CentraSiteCommand" provides support for removing an attribute, in cases where existing instances contain a value for the attribute. See below for details.
- You can add, modify, delete, rename and reorganize the profiles associated with a type at any time.

Important: If you are modifying one of the predefined asset types installed with CentraSite, review the information in *The Predefined Asset Types Installed with CentraSite* before you begin. It explains the kinds of modifications that you can make to the predefined types.

Important: If you are using CentraSite in conjunction with other software products, for example, the products of the Software AG webMethods Product Suite or a third-party product, those products can add their own asset types to CentraSite. Be aware that CentraSite treats these types as user-defined custom types, which can be modified by an administrator with the appropriate permissions (just like any other custom type). Modifying or deleting these types in CentraSite can lead to inconsistencies or errors in the product that uses the type. For example, if you modify or delete a type that is used by the webMethods Product Suite, components such as the webMethods Integration Server may no longer be able to publish assets to CentraSite. To prevent these types of errors, *do not modify or delete any asset type on which other Software AG components or third-party products depend*. For a list of the predefined types that the webMethods Product Suite uses, see *The Predefined Types Installed with CentraSite.*

Viewing or Editing a Type

To view or edit a type

1 In CentraSite Control, go to **Administration** > **Types**.

2 By default, all of the available types are displayed in the **Types** tab.

If you want to filter the list to see just a subset of the available types, enter a partial string in the **Search** field. CentraSite applies the filter to the **Name** column. The **Search** field is a type-ahead field, so as soon as you enter any characters, the display will be updated to show only those types whose name contains the specified characters. The wildcard character "%" is supported.

3 In the **Types** tab, click the name of the asset type that you want to modify.

4 In the Asset Type Details page, click **Edit** to open the **Edit Asset Type** wizard.

5 On panel 1, edit the following fields as necessary:

In this field...	Do the following...
Name	Specify the display name for the type. Be sure to use a name that your users will recognize and understand. For example, use "BPEL Process Document", not "bpdoc". The name you assign to the asset type can contain any character, including spaces. **Note:** Changing the type's display name will not affect its schema name. The type's schema name is fixed when a type is created. It is not affected by subsequent changes to the type's display name.
Description	*Optional*. Enter a brief description of the type.

6 If you want to modify the icons associated with the type or change the type's advanced settings, click **Advanced Settings** and modify the settings in the **Advanced Settings** dialog box as needed. For additional information about the settings in this dialog box, see the descriptions for these settings in *Creating a New Type*.

7 In panel 2, edit the type's attributes as necessary. If you need procedures for this step, see *Defining Attributes for a Type*.

8 In panel 3, edit the type's profiles as necessary. If you need procedures for this step, see *Defining Profiles for an Asset Type*.

9 In panel 4, change the selection of generic profiles as necessary. For more information about generic profiles, see *Generic Profiles*.

10 In panel 5, rearrange the order of the type's profiles as necessary.

11 Click **Finish** to save the updated type.

You can view multiple asset types as follows:

To view multiple asset types

1 In CentraSite Control, go to **Administration** > **Types**.

2 Ensure that the **Types** tab is selected.

3 Mark the checkboxes of the types whose details you want to view.

4 In the **Actions** menu, click **Details**.

The **Details** view of each of the selected types is now displayed.

Removing an Attribute

There is a command line tool that allows you to remove an attribute from an asset type, in cases where existing instances of the type contain a value for the attribute. Using the tool, the attribute's value is automatically removed from all existing instances, then the attribute is removed from the type.

Notes:

1. If there are many existing instances, the remove operation can take some time to complete.
2. You cannot remove a predefined attribute from a predefined asset type. You can, however, remove a custom (i.e. user-defined) attribute from a predefined asset type.

To remove an attribute from an asset type

- At the command line, enter a command of the following format:

```
CentraSiteCommand remove Attribute [-url <CENTRASITE-URL>] -user <USER-ID>
-password <PASSWORD> -assetType <ASSET-TYPE> [-attributeKind <ATTRIBUTE-KIND>]
-attributeName <ATTRIBUTE-NAME>
```

The following table describes the complete set of input parameters that you can use with the `remove Attribute` utility:

Parameter	Description
`-url`	The fully qualified URL (http://localhost:53307/CentraSite/CentraSite) for the CentraSite registry/repository.
`-user`	The user ID of a user who has the "CentraSite Administrator" role.
`-password`	The password of the user identified by the parameter "-user".
`-assetType`	The name of the asset type, in the format "{<namespace of the asset type>}SchemaName".
`-attributeKind`	A one-character code representing the type of the attribute you wish to remove. Allowable values are "C" for Classification, "R" for Relationship, "F" for File and "S" for all other attribute types. The use of the `attributeKind` parameter is optional. See the description of attribute types in the section ***What is a Type?*** for more information about the various kinds of attributes.
`-attributeName`	The name of the attribute schema for attributes whose *`AttributeKind`* is "S". For attributes with an *`AttributeKind`* other than "S" it is the name of the attribute itself.

Here is an example of a call of this command:

```
CentraSiteCommand remove Attribute [-url ↩
"http://localhost:53307/CentraSite/CentraSite"] -user "Administrator"
-password "manage" -assetType "{http://namespaces.CentraSite.com/Schema}XMLSchema" ↩
[-attributeKind "S"]
-attributeName "test_String_Attribute"
```

You can execute the above command in the command line interface *CentraSiteCommand.cmd* (Windows) or *CentraSiteCommand.sh* (UNIX) of Command Central. The tool is located in *<CentraSiteInstallDir>*/utilities.

If you start the 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".

6 Viewing or Editing a Virtual Type

Use the procedure in this section to view or edit a virtual type. When editing a predefined virtual type, keep the following points in mind:

- You cannot modify the type's base type attribute. This property is set when the type is created and cannot be changed thereafter.
- Virtual types inherit their attributes from their base type **Service** object. You cannot directly add attributes to, delete attributes from, or edit the properties of an attribute in a virtual type. To make these kind of modifications, you must edit the base type. For more information about virtual types, see *Virtual Types*.
- You can add, modify, delete, rename and reorganize the profiles associated with a virtual type at any time. However, you cannot modify or delete the profiles that are inherited from the type's base type.

To view or edit a virtual type

1 In CentraSite Control, go to **Administration** > **Types**.

2 By default, all of the available types are displayed in the **Types** tab.

If you want to filter the list to see just a subset of the available types, enter a partial string in the **Search** field. CentraSite applies the filter to the **Name** column. The **Search** field is a type-ahead field, so as soon as you enter any characters, the display will be updated to show only those types whose name contains the specified characters. The wildcard character "%" is supported.

3 In the **Types** tab, click the name of the virtual type that you want to view or edit.

4 If you want to view or edit the properties of the virtual type, click **Edit** and modify the basic attributes in the **Edit Asset Type** wizard as required.

5 If you want to view or edit the Inherit Base Type Profiles, Inherit Base Type Policies and Inherit Base Type LCM options, click **Advanced Settings** and modify the settings in the **Advanced**

Settings dialog box as needed. For additional information about the settings in this dialog box, see the descriptions for these settings in ***Creating a New Type***.

6 If you want to add or edit the profiles for the virtual type, see ***Defining Profiles for an Asset Type***.

7 Click **Finish** to save the updated type.

7 The Predefined Asset Types in CentraSite

CentraSite is installed with a number of predefined asset types. Some of these types are "core types" that belong to CentraSite itself. You can modify these types as described in ***Modifications You Can Make to CentraSite's Core Asset Types***.

Other predefined types are installed to support the use of CentraSite by products such as the webMethods Product Suite. These types belong to other products, which expect the type definitions to remain unchanged. Modifying or deleting these types in CentraSite can lead to inconsistencies or errors in the product that uses the type. For example, if you modify or delete a type that is used by the webMethods Product Suite, components such as the webMethods Integration Server may no longer be able to publish assets to CentraSite. You must not modify these predefined asset types.

The table in ***The Predefined Types Installed with CentraSite*** indicates which of the predefined types are core types that belong to CentraSite and which belong to other products such as the webMethods Product Suite.

The Predefined Asset Types Installed with CentraSite

The following table identifies the predefined asset types that are installed with CentraSite and indicates to which product they belong.

- The types that belong to CentraSite are core types that you can customize as described in ***Modifications You Can Make to CentraSite's Core Asset Types***. You cannot delete these types.
- The types that belong to the webMethods Product Suite are custom types that are installed with CentraSite. You *must not* modify or delete these types.

Type Name	Owner
Application	CentraSite
Application Server	CentraSite
BPEL Partner	CentraSite
BPEL Partner Link	CentraSite
BPEL Partner Link Type	CentraSite
BPEL Process	CentraSite
BPEL Role	CentraSite
BPM Process Project	webMethods Product Suite
CAF Security Role	webMethods Product Suite
CAF Task Rule	webMethods Product Suite
CAF Task Type	webMethods Product Suite
Decision Entity	webMethods Product Suite
E-form	webMethods Product Suite

Type Name	Owner
Event Type	webMethods Product Suite
Interface	CentraSite
IS Connection	webMethods Product Suite
IS Package	webMethods Product Suite
IS Routing Rule	webMethods Product Suite
IS Server	webMethods Product Suite
IS Service	webMethods Product Suite
IS Service Interface	webMethods Product Suite
IS Specification	webMethods Product Suite
IS Type Definition	webMethods Product Suite
JDBC Datasource	webMethods Product Suite
Operation	CentraSite
Package	CentraSite
Portlet	webMethods Product Suite
Portlet Preference	webMethods Product Suite
Process	webMethods Product Suite
Process Pool	webMethods Product Suite
Process Step	webMethods Product Suite
Process Swimlane	webMethods Product Suite
REST Service (virtual type of Service)	CentraSite
Rule Action	webMethods Product Suite
Rule Data Model	webMethods Product Suite
Rule Parameter	webMethods Product Suite
Rule Project	webMethods Product Suite
Rule Set	webMethods Product Suite
Service	CentraSite
TN Document Type	webMethods Product Suite
TN Group	webMethods Product Suite
Virtual REST Service (virtual type of Service)	CentraSite
Virtual Service (virtual type of Service)	CentraSite
Virtual XML Service (virtual type of Service)	CentraSite
Web Application	webMethods Product Suite
Web Application Page	webMethods Product Suite
WS-Policy	CentraSite
XML Schema	CentraSite
XML Service (virtual type of Service)	CentraSite

Modifications You Can Make to CentraSite's Core Asset Types

The following describes the ways in which you can customize the core asset types that belong to CentraSite.

- You can modify the type's existing properties and options (other than **Schema Name**, **Namespace** and **Base Type** (for virtual types only)).

  Note: Although CentraSite allows you to change the display name of the predefined types, we recommend that you do not do this. Name changes might lead to problems with future upgrades of CentraSite.

- You can add custom attributes to any type other than the predefined virtual types.
- You can edit the options for existing attributes of any type other than the predefined virtual types.
- You can add profiles, delete profiles, edit profiles and rearrange the order of profiles in the type.
- You *cannot* delete any of the predefined attributes that belong to the type. You can, however, delete custom (i.e. user-defined) attributes that belong to the type.
- You *cannot* modify the inherited profiles and attributes of the predefined virtual types.

8 Customizing the User and Organization Types

You can add custom attributes to the User and Organization objects that are installed with CentraSite. Adding custom attributes to these types enables you to include additional metadata about the organizations or users at your site. For example, if the organizations within your enterprise belong to specific affiliates, you might want the Organization object to include an attribute that identifies the affiliate to which an organization belongs.

You can add any number of custom attributes to a User or Organization. The attributes can be of any attribute type. You can also customize the icons that are associated with these types.

When you add custom attributes to a User or Organization type, the attributes appear on the **Attributes** tab when a user or organization is displayed in CentraSite Control.

To customize the Organization or User type definition

1 In CentraSite Control, go to **Administration** > **Types**.

2 In the **Types** tab, click the Organization type or the User type, depending on which type you want to customize.

3 In the Asset Type Details page, click **Edit** to open the **Edit Asset Type** wizard.

4 In panel 1, edit the following fields if you want to use a custom icon to represent this type in the user interface.

In this field...	Do the following...
Large Icon	*Optional*. Specify the large icon that is to be used to represent this type.CentraSite Control uses this icon when it displays the details page for instances of the type. If you want to use a custom icon, click the **Browse** button and upload the file containing the large version of the icon to CentraSite. The icon you use must be in GIF format. To ensure proper alignment when the icon is displayed in the user interface, it must be 64 x 64 pixels in size.

In this field...	Do the following...
Small Icon	*Optional*. Specify the small icon that is to be used to represent this type.CentraSite Control displays this icon when instances of this type appear in lists or summary tables. If you want to use a custom icon, click the **Browse** button and upload the file containing the small version of the icon to CentraSite. The icon you use must be in GIF format. To ensure proper alignment when the icons is displayed in the user interface, it must be 16 x 16 pixels in size.

5 In panel 2, edit the type's attributes as necessary. If you need procedures for this step, see *Defining Attributes for a Type*.

6 Click **Finish** to save the updated type.

9 Deleting a Type

When you delete an asset type, keep the following points in mind:

- You can delete a type *only if there are no instances of that type in the registry*.
- The core asset types that belong to CentraSite are non-deletable. CentraSite will not allow you to delete these types, even if there are no instances of the selected type in the registry. For a list of the core types that belong to CentraSite, see ***The Predefined Types Installed with CentraSite.***

Important: If you are using CentraSite in conjunction with other software products, for example, the products of the Software AG webMethods Product Suite or a third-party product, those products can add their own asset types to CentraSite. Be aware that CentraSite treats these types as user-defined custom types, which can be deleted by an administrator with the appropriate permissions (like any custom type). Deleting these types in CentraSite can lead to inconsistencies or errors in the product that uses the type. For example, if you delete a type that is used by the webMethods Product Suite, components such as the webMethods Integration Server may no longer be able to publish assets to CentraSite. To prevent these types of errors, *do not delete any asset type on which other Software AG components or third-party products depend*. For a list of the predefined types that the webMethods Product Suite uses, see ***The Predefined Types Installed with CentraSite.***

To delete an asset type

1 In the CentraSite Control, go to **Administration > Types** to display the asset types list.

2 Enable the checkbox next to the name of an asset type that you want to delete.

3 Click **Delete**.

When you are prompted to confirm the delete operation, click **OK**.

You can delete multiple asset types in a single step. The rules described above for deleting a single asset type apply also when deleting multiple asset types.

Important: If you have selected several asset types where one or more of them are predefined types, you can use the **Delete** button to delete the types. However, as you are not allowed to delete predefined asset types, only types you have permission for will be deleted. The same applies to any other types for which you do not have the required permission.

To delete multiple asset types in a single operation

1 In CentraSite Control, go to **Administration > Types** to display the asset types list.

2 Mark the checkboxes of the asset types that you want to delete.

3 From the **Actions** menu, choose **Delete**.

When you are prompted to confirm the delete operation, click **OK**.

10 Working with Composite Types

Overview

Certain assets can be stored in CentraSite as a set of related registry objects. Such assets are called composite assets. For example, if a web service provides several operations, this is stored in CentraSite as a composite asset consisting of the Service asset plus a separate Operation object for each of the web service's operations.

The objects that are constituents of a composite asset are referred to as components. In a composite asset there is a root component and one or more sub-components that are related to the root component. In the above example, the Service asset is the root component and the Operation objects are the sub-components. A sub-component of a composite asset can itself be a composite asset.

Depending on the relationships defined, registry operations (such as deleting an asset or exporting an asset) performed on a component of a composite asset can cause the same operation to be performed automatically on other components of the composite asset.

The concept of relationships between different objects in a SOA environment follows the UML idea of association relationships. This is only one of several forms of relationship supported by UML, but most SOA Registry Repositories only offer this form. CentraSite extends this scope to provide "aggregation" and "composition" relationships in addition to the existing association relationships. Each of these relationship forms provides its own semantics that affect specific operations that can be performed on composite assets.

You can define composite assets for all asset types, including custom (i.e. user-defined) asset types.

The CentraSite data model provides a means of representing composite assets, and allows operations to be performed on the entire composite asset or on sub-components in a consistent and well-defined manner.

The following operations take the composition definitions into account:

- Deleting an asset
- Exporting an asset
- Creating a new version of an asset
- Setting the instance permissions on an asset
- Changing the owner of an asset
- Moving an asset to another organization

Note: Lifecycle state propagation is not included in the above list, as experience has shown that such models can cause major problems in their definition and consistency rules. If such a model is required, then it should be implemented via a custom pre/post-state change policy.

Shared vs Nonshared Components

Sometimes a component can serve as a constituent of multiple composite objects. For example, XML schema "ABC" might contain schema "XYZ" as one of its components. Other services and/or schemas might also include schema XYZ as a component. Components that can belong to more than one composite object are referred to as *shared components*. Components that can only belong to a particular instance of a composite object are referred to as *nonshared components*. For example, the operations, bindings and interfaces associated with a Web service are considered *nonshared* components. These objects belong solely to the service and cannot function as constituents of other composite objects. Schemas, however, are considered *sharable*, meaning that they do not belong exclusively to a particular composite object.

Required Objects

Besides components, a composite object can also have *required objects*. Required objects are registry objects and/or repository items that are not actually part of the composite object itself, but support or augment the composite object in an essential way. For example, if a Service object has a WS-Policy attachment, the attached policy is treated as a required object because it specifies the WS-Policies that must be applied to the service when it is deployed.

Required objects, while not actually part of the composite object, must be present in the registry to make the object wholly complete or usable. (An asset's required objects are generally objects that the export process must bundle with the asset in order for the asset to be wholly represented and functional in another registry.)

Collectors

A collector is an internal process within CentraSite that identifies all of the constituents of a composite object. A collector examines a given object and returns lists that identify:

- The nonshared components associated with the composite object
- The shared components associated with the composite object
- The required objects associated with the composite object

Each composite type has its own collector. The lists produced by a collector are used by handlers that operate on instances of composite objects. For example, when you delete an XML Schema, the delete handler for schemas deletes the schema itself and all of the schema's nonshared components as identified by the collector for XML Schemas.

Defining Composite Asset Types

The relationships between components of a composite asset are defined using relationship attributes available in the appropriate asset type definition(s). A relationship can be defined on the root component or on a sub-component.

In addition to using the predefined composite asset types, you can define your own composite asset types. A user-defined composite asset type consists of the following parts:

- a user-defined asset type; each instance of this type will be the root component of a composite asset, and
- other asset types or object types; instances of these types will be related to the root component or to each other by means of relationship attributes.

The definition of a relationship may be changed at any time without affecting any instances.

You set up the associations between the components of a composite asset by using attributes of the data type "Relationship" in the asset type definition. A relationship indicates a coupling between two objects. A relationship has a direction, meaning that one of the related objects is the source of the relationship and the other object is the target of the relationship. When you define a relationship, you define it on the source object, not on the target object.

See the section **Creating a New Type** for information on how to create attributes of the data type "Relationship".

The semantic of a relationship is usually indicated by the name you choose for the association type of the relationship attribute (e.g. "hasChild" or "hasParent"). You can think of the association type as a label that does not affect the behavior of the composite asset (technically it is a classification on the relationship attribute), although it makes sense to choose meaningful association types for the relationship attributes. To inform CentraSite about the semantics of the associations in your composite asset type, you need to define the relationship attributes.

CentraSite provides several forms of relationship that allow you to define plain relationships between assets as well as relationships for composite assets.

For our purposes, we will use terms and concepts introduced by UML as follows:

- **Association**
  The loosest form of coupling is provided by the *association* relationship. This is like a cross-reference between two components. It indicates that there is a dependency between the components but no aggregation or composition. In this case, registry operations performed on a component do not cause any operation to be performed automatically on the related component. For example, suppose Asset A contains an association relationship to Asset B, and then Asset A is then deleted; in this case, the registry remains in a consistent state without having to delete or modify Asset B in any way.

NOTE: when an asset instance has an incoming relationship it may not be deleted until that incoming relationship has been removed or the asset that is the source of the relationship is in the delete set.

- **Aggregation**
  A tighter coupling is provided by the *aggregation* relationship. Aggregation is similar to a whole/part relationship in which components of a structure can also exist independently of the structure; this is like the "contains" semantic, whereby one component contains another component but does not own it. In this case, some operations performed on a component cause the same operation to be performed automatically on the related component. For example, if you want to export an asset, CentraSite automatically extends the export set by adding all of the components that are coupled by aggregation. However, if you want to delete an asset, CentraSite leaves the coupled components unchanged.
- **Composition**
  The tightest coupling is provided by the *composition* relationship. Composition is similar to a whole/part relationship in which components of a structure cannot exist independently of the structure; this is like the "owns" semantic, whereby one component owns another component. In this case, all registry operations performed on a component cause the same operation to be performed automatically on the related components. For example, if you want to delete an asset, then CentraSite automatically extends the delete set by adding all of the components that are coupled by composition.

The form of relationship determines the way in which registry operations performed on one component affect the related components. In the following table, entries marked with "yes" mean that an operation on a component causes the same operation to be performed on the related components, whereas table entries marked with "no" mean that the related components are not changed.

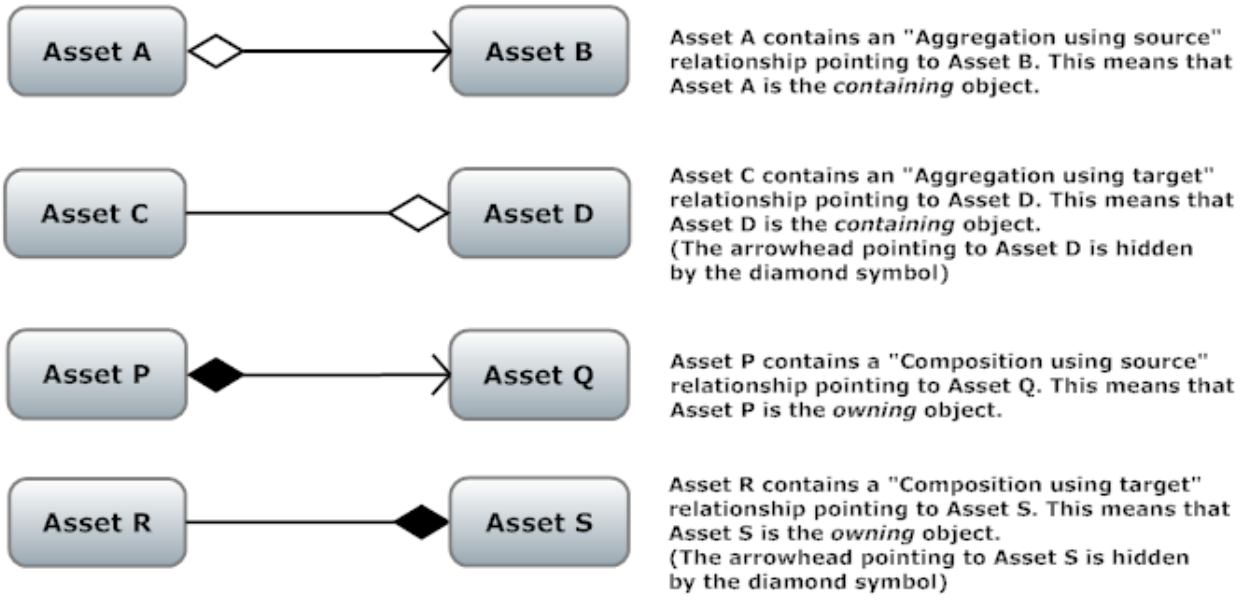
Operation on component	Form of Relationship:		
	Association	Aggregation	Composition
Move asset to another organization	no	no	yes
Change asset owner	no	no	yes
Delete asset	no	no	yes
Export asset	no	yes	yes
Set instance permissions on an asset	no	yes	yes
Create new version of an asset	no	no	yes

These operations are the primary set which are affected by different forms of relationships and are supported out-of-the-box by CentraSite.

Aggregation and Composition come in two forms, namely "with source" and "with target":

- **Aggregation/Composition with source**
  This means that the aggregation or composition treats the source component (i.e. the component where the relationship is defined) as the containing or owning component, and the target component (i.e. the component that the relationship points to) as the contained or owned component.
- **Aggregation/Composition with target**
  This means that the aggregation or composition treats the source component (i.e. the component where the relationship is defined) as the contained or owned component, and the target component (i.e. the component that the relationship points to) as the containing or owning component.

You might find the following diagrams useful to illustrate the relationships in composite assets. They are similar to UML diagrams, but allow the aggregation or composition to be on the target component (in UML, they can only be on the source component). The forms "with source" and "with target" are represented using a diamond-shaped symbol to indicate the containing/owning component. Aggregation is indicated by a non-filled diamond symbol and Composition is indicated by a filled diamond symbol. An arrow points from the source component to the target component, with the arrowhead located at the target component. If the diamond symbol and the arrowhead are located at the same component, only the diamond is shown.



Semantics of Relationships and Operations

Association Relationship

Association relationships are the relationships that were available in later releases of CentraSite v8 and are available with unchanged semantics in the current release.

Aggregation Relationship

The aggregation relationship changes the rules in the following way for operations:

Operation	Rules
Delete an asset	There are no changes in the delete rules when introducing aggregation.
Create a new version of an asset	There are no changes in the versioning rules when introducing aggregation. However, for assets of type "Service" and "XML Schema", there is an additional possibility: If you mark the checkbox "Propagate to dependent objects" when you create a new version of the root component of a composite asset of one of these types, the versioning will be propagated also to components of these types that are connected to the root component via aggregation relationships.
Set instance permissions on an asset	Merges the permissions of the initiating component with those of the current component. The permissions assigned to the contained component are the union of the permissions of the containing asset and the contained component. If the user that performs the operation does not have Full permissions on a component, then it and all of its sub-components will be skipped.
Move asset to another organization	There are no changes in the move organization rules when introducing aggregation.
Change the owner of an asset	There are no changes in the change owner rules when introducing aggregation.
Export an asset	If a component that has a *containing* aggregation is added to the export set, then the target is also added to the export set. The rules when selecting the checkbox 'including instances' in the user interface apply as before with the addition of the containing rules.

Note: For Export, the usage of recursive relationships on the type and instance level must be taken into account. Whereby type level does not mean that the same instance is referenced.

Composition Relationship

Composition relationships affect all of the defined operations to varying degrees.

Operation: Deleting an asset

On deletion, if the root component is added to the set to be deleted, then the sub-components will also be added to the set to be deleted. The direction of the association does not play a role in defining the set, only the *containing* designation. This means it is possible for the deletion to fail if one of the assets added to the deletion set during this processing is referenced via the basic association relationship target rules.

This rule is applied recursively. For example, if we have three assets that have the relationships "A contains B contains C", then the following statements apply:

- When A is deleted, then B will be deleted and finally because B is deleted, C will also be deleted.
- When deleting C, only C will be deleted.

This fails if the deleting user does not have permission on any of the assets in the set acquired by traversing the graph. The delete is considered atomic - either all are deleted or none. This avoids inconsistencies in the outcome of the operation.

The relationship direction always plays a role in the deletion operation. An asset may not be deleted if it is the target of a relationship and the source is not part of the deletion set.

The deletion rules described here apply also when you purge old versions of an asset. In this case, the purge operation will be applied not only to the component being purged, but also to the related sub-components.

Operation: Creating a new version of an asset

On versioning, if the root component is added to the set to be versioned, then the sub-components will also be added to the set. The direction of the association does not play a role in defining the set, only the *containing* designation.

If you create a new version of an asset that is the root component of a composite asset, and the root component is related to one or more of the other components via composition relationships, CentraSite automatically creates a new version of each of these other components.

Operation: Exporting an asset

On export, if the root component is added to the set to be exported, then the sub-components will also be added to the set. The direction of the association does not play a role in defining the set, only the *containing* designation.

Operation: Setting instance permissions on an asset

On setting permissions, when the root component is added to the set to which the permissions will be applied, then the sub-components asset are also added if and only if the user has the permission to modify the permission of the target. If the user does not have permission, then the graph traversal for the target is not carried further for this sub-graph.

The permission set that will be given to all sub-component assets is the merge based on what is to be modified. The permissions assigned to the owned asset are the union of the permissions of the owning asset and the owned asset.

Operation: Moving an asset to another organization

On moving an asset to another organization, when the root component is added to the set to be moved, then the sub-components are also added. No permission checks are done during this operation as only users in the CentraSite Administrator role may perform this operation.

Operation: Changing the owner of an asset

On changing ownership, when the root component is added to the set to be changed, then the sub-components are also added to the set. No permission checks are done during this operation as only users in the CentraSite Administrator role may perform this operation.

Extended Rules

This section deals with the extended rules or semantics that have been added to Aggregation and Composition relationships.

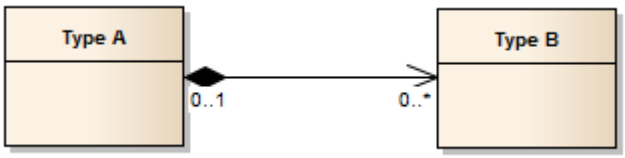
Changing Relationships

As part of the support for Aggregation and Composition, CentraSite allows the relationship form to be changed after the type is created. This change affects all current instances and new instances. This means that after a relationship attribute is created, the form (for association the default form, for aggregation (both forms, i.e. "using source" and "using target") and for composite (both forms)) can be changed by an Asset Type Administrator. From that point onwards, the appropriate rules will be applied when performing the defined operations.

Updating Assets

The following asset updates need to be taken into account when implementing models:

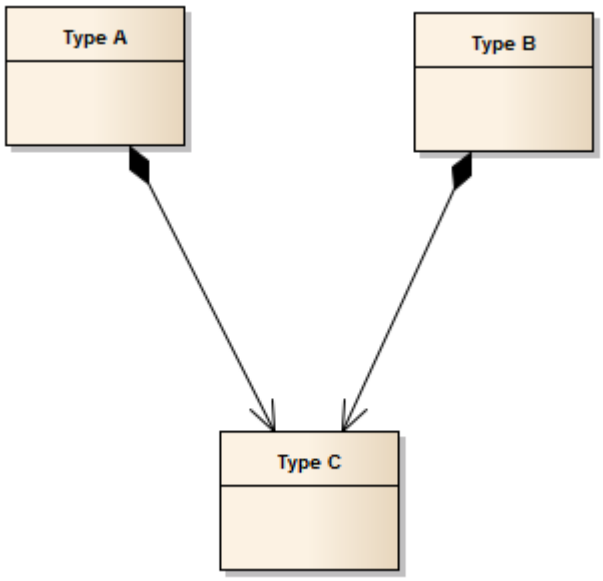
1. Adding relationships to existing instances. Given the below model:



   It is perfectly legal to create instances of Type A and Type B independent of one another. In fact in CentraSite this characteristic is mandatory, as the creation of multiple assets at the same time is only allowed in a few places in the UI.

   When adding the relationship between an instance of Type A and an instance of Type B, CentraSite will not do any extra operations to guarantee the consistency of permissions at this point.

2. Adding relationships to 2 different composites. Given the below model (which is a legal model):



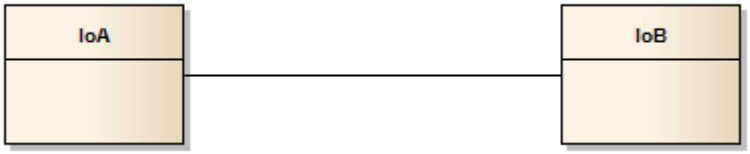
The following restriction applies to user-defined asset types, but not predefined asset types: At runtime if an instance of Type A creates a composite relationship to an instance of Type C, and then an instance of Type B tries to create a composite relationship to the same instance of Type C, this composition will be rejected. This is because a contained asset (instance of Type C) can only have one owning asset (instance of Type A or instance of Type B).

Usage Scenarios

Overview

In the following sections, the outcome of each operation is given based on a very simple type and instance configuration.

Unless otherwise stated, the instances that each operation will be performed on will be:



Key	Description
IoA	Instance of type A
IoB	Instance of type B

Delete usage scenarios

Association Relationship

Given the type model:

The result of the delete operation will be:

Operation	Expected result
Delete IoB	Fail because of incoming relationship from IoA
Delete IoA	Success. Post-condition: IoB will be left intact

Aggregation Relationship with containing constraint on Type A

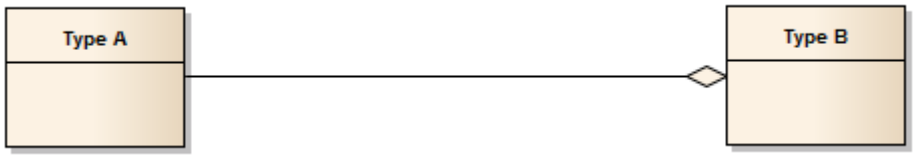
Given the type model:

The result of the delete operation will be:

Operation	Expected result
Delete IoB	Fail because of incoming relationship from IoA
Delete IoA	Success. Post-condition: IoB will be left intact

Aggregation Relationship with containing constraint on Type B

Given the type model:

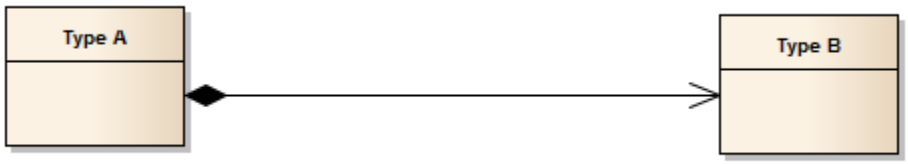


The result of the delete operation will be:

Operation	Expected result
Delete IoB	Fail because of incoming relationship from IoA
Delete IoA	Success. Post-condition: IoB will be left intact

Composition Relationship with containing constraint on Type A

Given the type model:



The result of the delete operation will be:

Operation	Expected result
Delete IoB	Fail because of incoming relationship from IoA
Delete IoA	Success. Post-condition: IoB will be removed

Composition Relationship with containing constraint on Type B

Given the type model:

The result of the delete operation will be:

Operation	Expected result
Delete IoB	Success. Post-condition: IoA will be removed
Delete IoA	Success. Post-condition: IoB will be left intact

Composition Relationship with permission scenario

Given the type model:

With the constraints:

- User who will perform the deletion is Fred
- Fred has Full permission on IoA
- Fred has Read permission on IoB

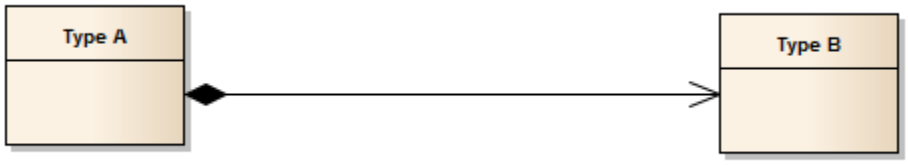
The result of the delete operation will be:

Operation	Expected result
Delete IoB	Fail. User Fred does not have full permission on IoB
Delete IoA	Fail. User Fred does not have full permission on IoB

Versioning usage scenarios

Versioning for Association Relationship and Aggregation Relationship are the same and do not change from previous versions, therefore only Composition Relationship are shown below.

Given the type model:

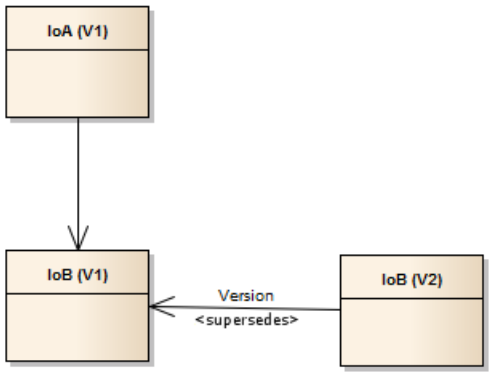


Note: Both variants of a composite relationship (source and target) are supported and are orthogonal.

Based on the instances given above, the following scenarios are considered relevant.

Versioning of IoB

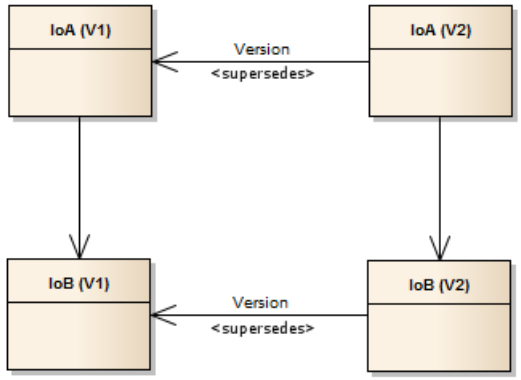
This causes just IoB to be versioned, and the IoA version is left unchanged. Pictorially, this looks like:



Versioning of IoA

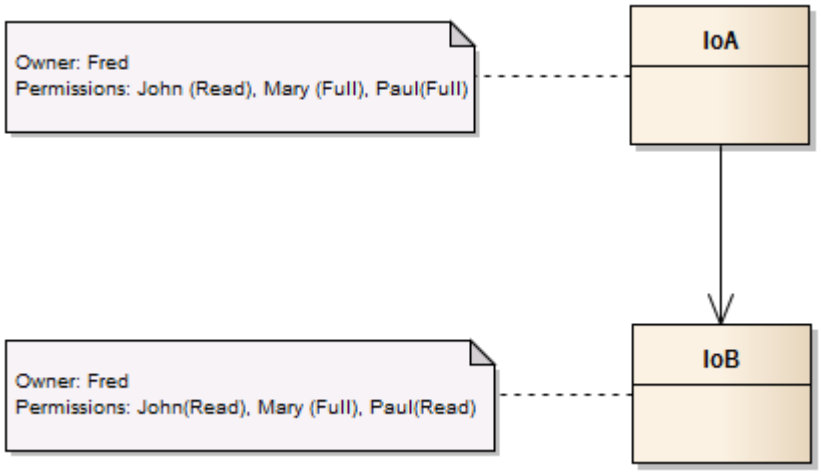
Versioning of IoA will result in the composite relationship being used to work out which other assets should be versioned at the same time. This will result in IoA and IoB being versioned together (if we fail to version either then neither will be versioned).

This pictorially looks like:



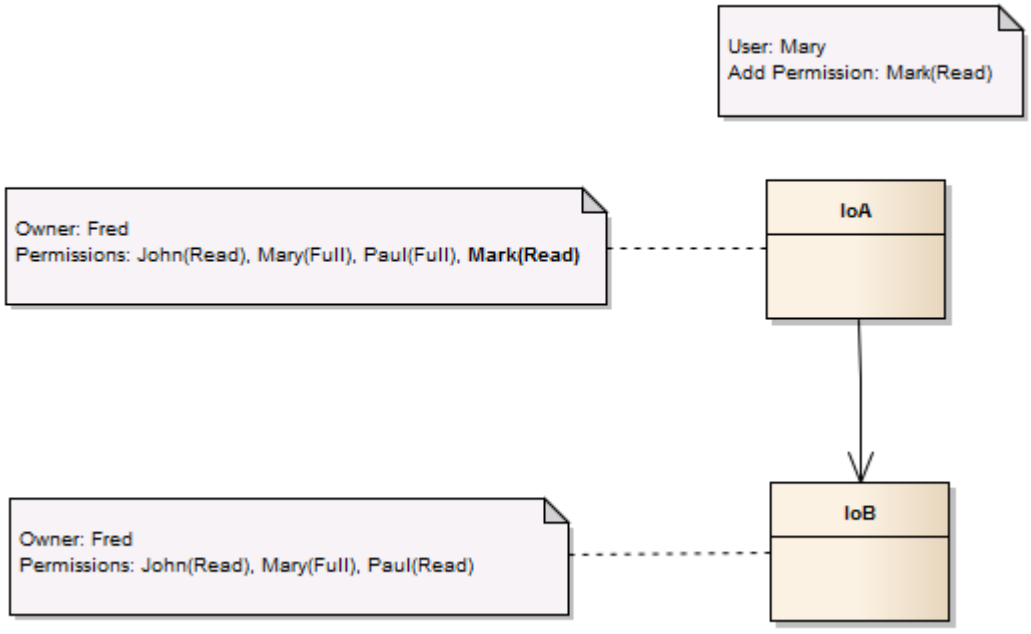
Permission usage scenarios

For the permission scenarios, the following instances with the annotations for the owner and permissions will be used as basis.



Association Relationship permission propagation

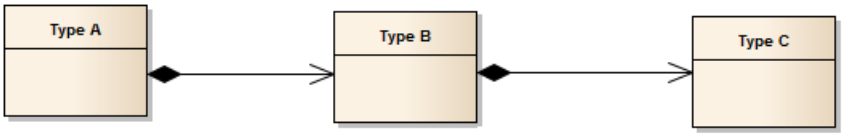
When an Association Relationship is used, the permission propagation does not take place. This means that if an instance permission is set, then only that instance's permission is affected. This means that if user Mary adds Read permission for user Mark on IoA, then Mark only gets permission to Read IoA. He does not get permission to Read IoB. Pictorially this looks like:



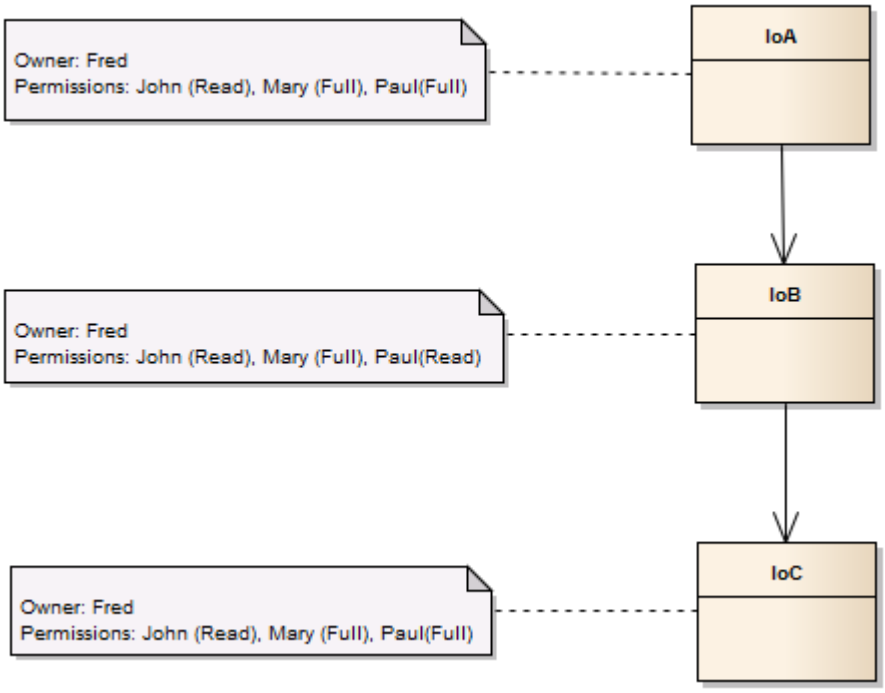
Composition/Aggregation with weak propagation

One of the key points of permission propagation is what happens if a user only has a Full permission on a subset of the assets to which the permissions need to be propagated. In this case, the usage of the so-called weak propagation rule comes into effect. The rule states that if a user does not have permission to propagate to all instances in the set, then the permissions will be propagated to only the instances which are allowed.

For this scenario the following model will be used:

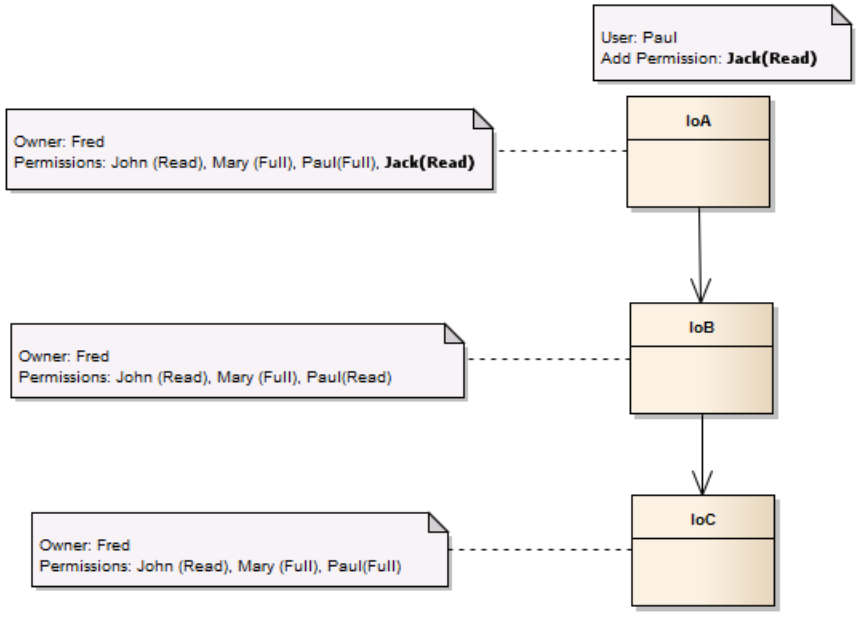


Based on this model, the instances and the permissions to start with should be:



Now if user Paul adds Read permission for user Jack to IoA, Jack will only get this permission on IoA as Paul does not have the rights to give Jack the permissions on IoB. Even though Paul has Full permission on IoC, because it is a child of IoB, Jack does not get the permissions for IoC because of weak propagation. We trim/terminate the propagation at IoB.

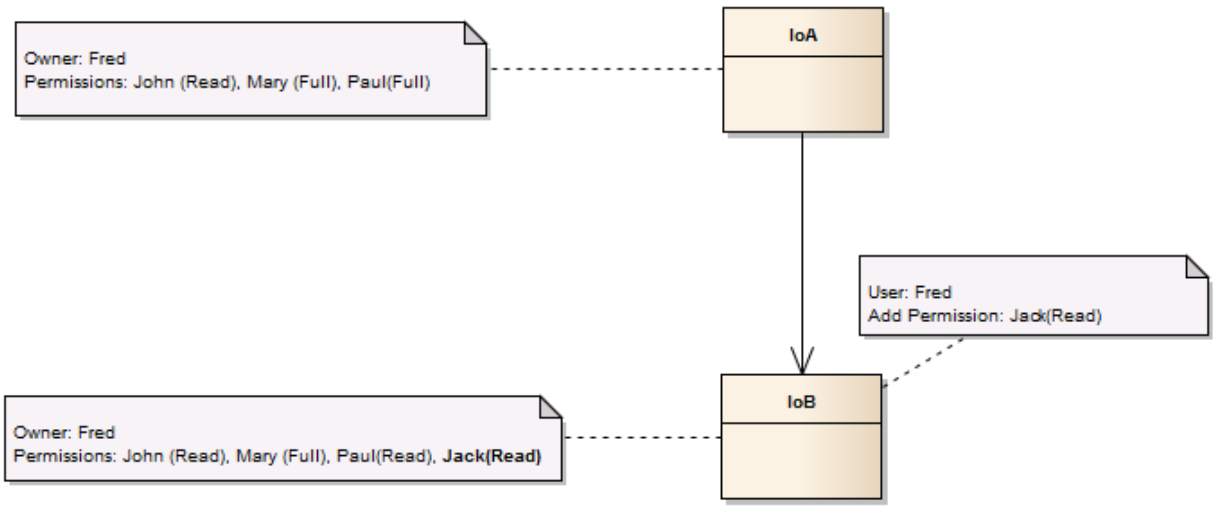
This pictorially looks like:



Composition/Aggregation Relationship updating sub-component

Updating the permissions of a sub-component without affecting the overall composition/aggregation is not affected with the changes. Therefore if user Fred wants to explicitly add Read permission for Jack on IoB, this is possible.

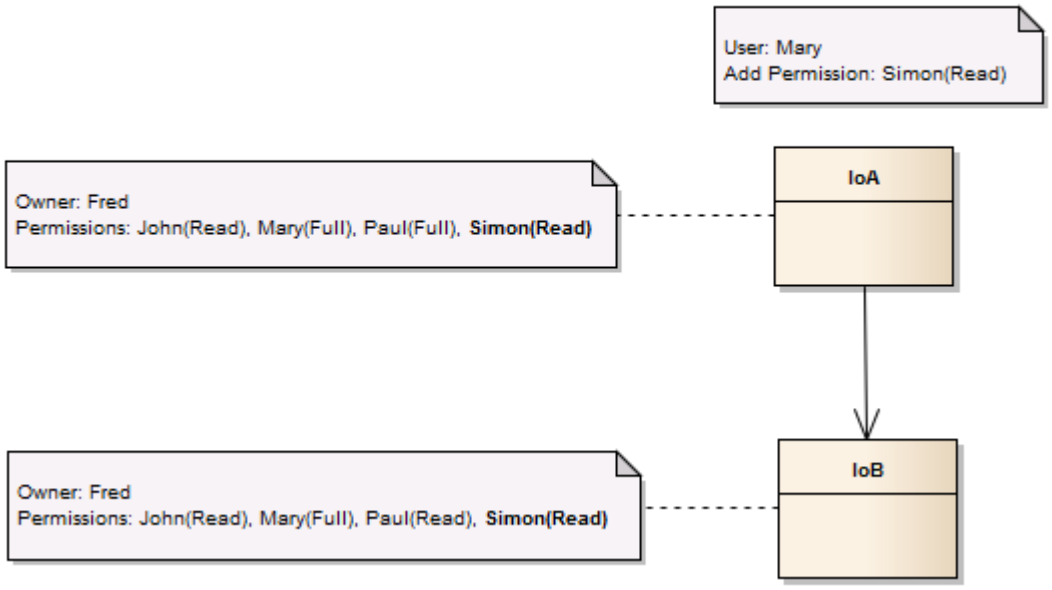
This pictorially looks like:



Composition/Aggregation Relationship with full propagation

Full propagation happens when all sub-components can be updated by the instigating user. As example, Mary wants to give Read permission to Simon on IoA with permission propagation via the Composition/Aggregation relationship. As Mary has Full permissions on IoA and IoB, the permissions are propagated over the relationship.

This pictorially results in:

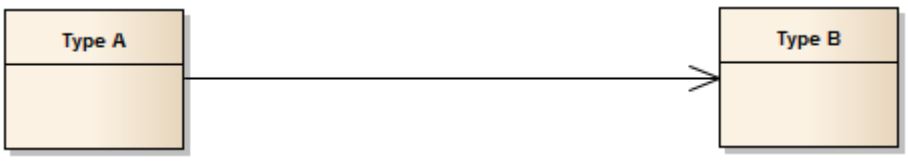


Export usage scenarios

Export with Association Relationship

Export will work the same as in previous versions - the usage given here assumes that no additional options are chosen.

Given the model:



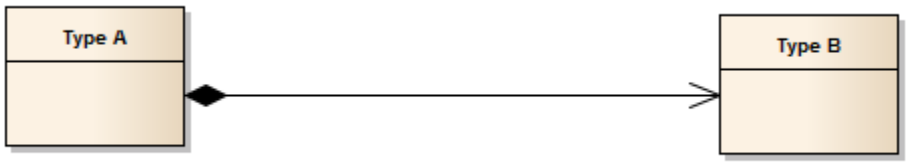
The result of the Export operation will be:

Operation	Expected result
Export IoB	Export set contains IoB. It does not contain IoA
Export IoA	Export set contains IoA. It does not contain IoB

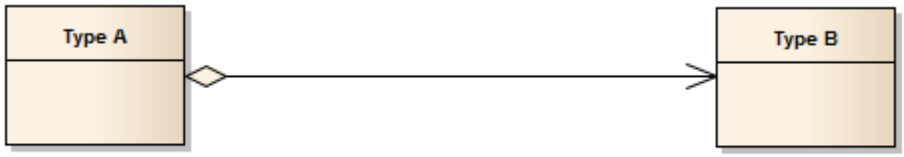
Export with Composition/Aggregation Relationship

For both Composition and Aggregation Relationships, the rules are exactly the same. When such a Relationship with the appropriate containing rule is found, then traverse the relationship and add sub-components to the set.

Given the model:



OR



The result of the Export operation will be:

Operation	Expected result
Export IoB	Export set contains IoB. It does not contain IoA
Export IoA	Export set contains IoA and IoB.

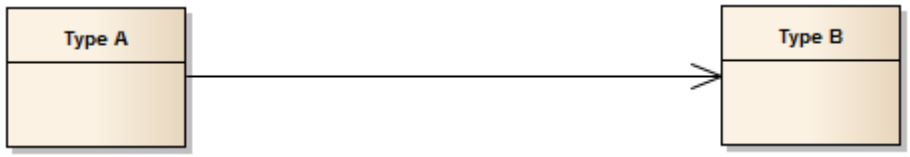
Move Organization usage scenarios

Move organization is an administrative task and may only be performed by someone with appropriate administration rights. This means that permissions and ownership do not play a role when performing the move operation.

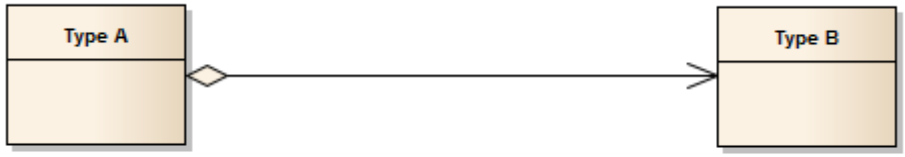
Move Organization with Association/Aggregation Relationship

When moving an asset from one organization to another, the Association and Aggregation Relationships do not change any related assets. This is because both of these relationships are considered to be loosely coupled.

Given the model:



OR



The result of the Move Organization operation will be:

Operation	Expected result
Move IoB	Only IoB will be moved to the new organization. It does not move IoA.
Move IoA	Only IoA will be moved to the new organization. It does not move IoB.

Move Organization with Composition Relationship

When a Composition Relationship with the appropriate containing rule is found, then traverse the relationship and move the sub-components to the new organization.

Given the model:

The result of the Move Organization operation will be:

Operation	Expected result
Move IoB	Only IoB will be moved to the new organization. It does not move IoA.
Move IoA	IoA and IoB will be moved to the new organization.

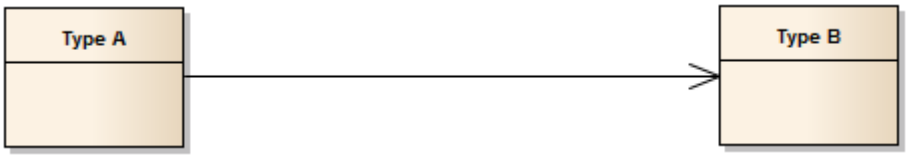
Change Ownership usage scenarios

Change Ownership is an administrative task and may only be performed by someone with appropriate administration rights. This means that permissions and ownership do not play a role when performing the change ownership operation.

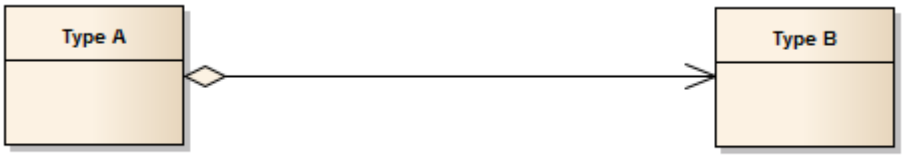
Change Ownership with Association/Aggregation Relationship

When changing an asset's ownership from one user to another, the Association and Aggregation Relationships do not change any related assets. This is because both of these relationships are considered to be loosely coupled.

Given the model:



OR



The result of the Change Ownership operation will be:

Operation	Expected result
Change Ownership of IoB	Change the ownership of IoB. It does not affect IoA.
Change Ownership of IoA	Change the ownership of IoA. It does not affect IoB.

Propagation of Profile Permissions

In addition to propagating permissions that control the access to an asset instance (as described above), it is also possible to propagate permissions that control the access to the asset instance's profiles.

Profile permissions of the root asset of a composite asset can be propagated to the other components if the components have the same type as the root asset. This restriction arises because different asset types can have different sets of profiles, whereas assets of the same type have the same set of profiles.

Propagation of profile permissions is activated when you mark the checkbox **Propagate profile permissions** in the asset's **Permissions** tab. This checkbox can only be selected if you have also marked the checkbox **Propagate Permissions to dependent objects**.

The Predefined Composite Asset Types

The following identifies the nonshared components, shared components, and required objects that are associated with each of the predefined composite types installed with CentraSite.

- Service
- Virtual Service
- XML/REST Service
- Virtual XML/REST Service
- XML Schema
- BPEL Process
- WS-Policy
- BPM Process Project

- BPM Process Step
- IS Package
- IS Service Interface
- Process
- Process Pool
- Process Swimlane
- Web Application
- Portlet
- CAF Task Type
- TN Group
- Business Rules Project
- Data Model

Service

Nonshared Components	Description
Binding(s)	The objects that represent the specific ports that are defined in the WSDL. (A port defines a specific endpoint where the service is provided.)
Interface(s)	The objects that represent the portType that is a defined in the WSDL. (A portType defines a set of operations that the service provides.)
Operation(s)	The objects that represent the individual operations that the service provides.
Service WSDL	The ExternalLink to the WSDL file and the WSDL file itself.
Other WSDL (references to WSDLs that are imported or included in the main service WSDL)	The ExternalLinks to the referenced WSDL files and the referenced WSDL files themselves. **Important:** If, at collection time, the collector discovers that a referenced WSDL is also a component of another asset, it places that WSDL in the *shared component* list. Otherwise, it returns the referenced WSDL in the list of nonshared components.
BPEL Partner Link Type	The object that represents the BPEL Partner Link Type to which the service is related (if such an association exists).
BPEL Role	The object that represents the BPEL Role to which the service is related (if such an association exists).

Shared Components	Description
XML Schema(s)	The entire graph of XML schemas that are related to the service. (The graph includes all of the XML schemas that the service references directly or indirectly). For each XML schema in the graph, the collector collects the ExternalLink to the XSD file and the actual XSD file itself.

Required Objects	Description
Service type	The Type object that defines the structure of a Service object in this registry (including all user-defined profiles that have been defined for the type).
WS-Policy	The WS-Policy objects that are associated with the service (if any).
Supporting Documents	ExternalLinks that point to files in the supporting document library plus the files themselves (if any). **Note:** The list of supporting documents that the collector returns includes 1) documents that have been attached to the service using any of the predefined File attributes defined in the Service type, 2) documents that have been attached to the service using a custom File attribute, and 3) any documents that have been attached to the service using an ad-hoc ExternalLink.

Virtual Service

Nonshared Components	Description
The set of nonshared components defined for the Service type.	See nonshared components under **Service** above.
WS-Policy	The WS-Policy object associated with the virtual service.
Processing Steps	The policy objects that represent the processing steps for the virtual service.
Extrinsic object	An internal copy of the WSDL that is maintained for the virtual service.
VSD	The virtual service's virtual service descriptor (VSD).

Shared Components	Description
The set of shared components defined for the Service type.	See shared components under **Service** above.

Required Objects	Description
Service type	The Type object that defines the structure of a Service object in this registry (including all user-defined profiles that have been defined for the type).
Native service	The Service object from which the virtual service was generated.
Supporting Documents	ExternalLinks that point to files in the supporting document library plus the files themselves. **Note:** The list of supporting documents that the collector returns includes 1) documents that have been attached to the service using any of the predefined File attributes defined in the Service type, 2) documents that have been attached to the virtual service using a custom File attribute, and 3) any documents that have been attached to the virtual service using an ad-hoc ExternalLink.

XML/REST Service

Nonshared Components	Description
ServiceBinding(s)	Each ServiceBinding object represents an <endpoint/> element of the XML/REST Service WSDL20.
Binding Concept(s)	Each Binding Concept object represents a <binding/> element of the XML/REST Service WSDL20.
Interface(s)	Each Interface object represents an <interface/> element of the XML/REST Service WSDL20.
SpecificationLink(s)	SpecificationLinks are used to link the Interface and Binding Concept objects to a ServiceBinding object.
Operation(s)	Each Operation object represents an XML/REST Service Resource and is represented as an element in the WSDL20.
Service WSDL20	The ExternalLink to the WSDL20 file (in the CentraSite repository) and the WSDL20 file itself.

Shared Components	Description
XML Schema(s)	The entire graph of XML Schemas that are related to the XML/REST Service. (The graph includes all of the XML Schemas that the XML/REST Service's Resources reference directly or indirectly). For each XML Schema in the graph, the collector collects the ExternalLink to the XSD file (in the CentraSite repository) and the actual XSD file itself.

Required Objects	Description
Service Type	The ObjectType Concept that defines the structure of a Service object in this registry (including all user defined profiles that have been defined for the type).
CentraSiteVirtualType	The CentraSiteVirtualType Concept that identifies the VirtualType of the Service object.
WS-Policy	The WS-Policy objects that are associated with the service (if any).
Supporting Documents	ExternalLinks that point to files in the supporting document library plus the files themselves (if any). **Note:** The list of supporting documents that the collector returns includes 1) documents that have been attached to the service using any of the predefined File attributes defined in the Service type, 2) documents that have been attached to the service using a custom File attribute, and 3) any documents that have been attached to the service using an ad-hoc ExternalLink.

Virtual XML/REST Service

Nonshared Components	Description
The set of nonshared components defined for the XML/REST Service type.	See nonshared components under **XML/REST Service** above.
WS-Policy	The WS-Policy object associated with the virtual service.
Processing Steps	The policy objects that represent the processing steps for the Virtual XML/REST Service.
Extrinsic object	An internal copy of the WSDL20 that is maintained for the Virtual XML/REST Service.
VSD	The Virtual XML/REST Service's virtual service descriptor (VSD).

Shared Components	Description
The set of shared components defined for the XML/REST Service type.	See shared components under **XML/REST Service** above.

Required Objects	Description
Service type	The ObjectType Concept that defines the structure of a Service object in this registry (including all user-defined profiles that have been defined for the type).
CentraSiteVirtualType	The CentraSiteVirtualType Concept that identifies the VirtualType of the Service object.
Native service	The XML/REST Service object from which the Virtual XML/REST Service was generated.
Supporting Documents	ExternalLinks that point to files in the supporting document library plus the files themselves. **Note:** The list of supporting documents that the collector returns includes 1) documents that have been attached to the service using any of the predefined File attributes defined in the Service type, 2) documents that have been attached to the virtual service using a custom File attribute, and 3) any documents that have been attached to the virtual service using an ad-hoc ExternalLink.

XML Schema

Nonshared Components	Description
XSD File	The ExternalLink to the XSD file and the XSD file itself.
XML Schema(s) (referenced)	The entire graph of XML schemas that are related to this XML schema. (The graph includes all of the schemas that this XML schema references directly or indirectly). For each XML schema in the graph, the collector collects the ExternalLink to the XSD file and the actual XSD file itself.

Shared Objects	Description
None	n/a

Required Objects	Description
XML Schema type	The Type object that defines the structure of an XML Schema object in this registry (including all user-defined profiles that have been defined for the type).

BPEL Process

Nonshared Components	Description
BPEL File	The ExternalLink to the BPEL document and the BPEL document itself.
BPEL Partners	The objects that represent partners in the BPEL process.
PartnerLinks	The objects that represent partner links in the BPEL process.
Association Type (Service)	The association type that CentraSite uses to relate a BPEL process to a service.

Shared Objects	Description
None	n/a

Required Objects	Description
BPEL type	The Type object that defines the structure of a BPEL object in this registry (including all user-defined profiles that have been defined for the type).
Services	The services that are referenced by the BPEL's PartnerLinks. (This includes all of the components and required objects associated with the referenced services. For a detailed list of these components and required objects, see **Service**, above.)
PartnerLinkTypes	The PartnerLinkTypes that are referenced by the BPEL's PartnerLinks.
Roles	The Roles that are referenced by the BPEL's PartnerLinks.

WS-Policy

Nonshared Components	Description
Policy	The object representing the policy itself.
Policy Parameter	The objects which form the parameters defined for the policy (if any). **Note:** There can be multiple levels of parameters which can be nested; parameters from all levels will be included.
Policy Condition (if any)	The object representing the conditions defined for the policy (for example, if the policy has conditions such as "Name contains XML").

Shared Objects	Description
None	n/a

Required Objects	Description
Custom policy action	A custom action that is used by the policy and is not available as part of the predefined set.
Custom policy action parameters	All action parameters used by a custom action that the policy uses (if any).
Custom asset types	The custom defined asset types that the policy is defined for (if any).
Registry object	The registry object that the policy uses as a parameter (if any).

BPM Process Project

Components	Description
Process	Contains a set of Process Steps that invokes services and possibly other processes and has documents as inputs and outputs.

BPM Process Step

Components	Description
CAF Task Type	webMethods Task Engine human activity task.
CAF Security Role	A role which has the privilege to participate in CAF actions.
E-Form	The E-form object represents electronic forms that support forms-driven processes.
IS Service	Represents a webMethods IS Service Type.
Process Pool	Allows grouping of process steps into an internal or external process. More than one pool may exist within a process.
Process	Contains a set of Process Steps that invokes services and possibly other processes and has documents as inputs and outputs.
Service	A service is a software component that is described via a well defined interface and is capable of being accessed via standard network protocols such as, but not limited to, SOAP over HTTP. CentraSite is able to extract metadata of services based on a WSDL description.
User	The predefined JAXR-based type User.
XML Schema	A reference to an XML Schema file

IS Package

Components	Description
IS Specification	Represents a webMethods IS Specification Type.
IS Type Definition	Represents a webMethods IS Type Definition Type.
IS Routing Rule	Represents a webMethods IS Routing Rule Type.
IS Service Interface	Represents a webMethods IS Service Interface Type.
IS Connection	Represents a webMethods IS Connection Type.
IS Service	Represents a webMethods IS Service Type.

IS Service Interface

Components	Description
Binding(s)	The objects that represent the specific ports that are defined in the WSDL. (A port defines a specific endpoint where the service is provided.)
Interface(s)	The objects that represent the portType that is a defined in the WSDL. (A portType defines a set of operations that the service provides.)
Operation(s)	The objects that represent the individual operations that the service provides.
IS Service	Represents an Integration Server (IS) Service operation.
REST Service	Represents a corresponding REST Service in the Integration Server (IS).

Process

Components	Description
Process Step	Represents an activity in a Process.
Process Pool	Allows grouping of Process Steps into an internal or external Process. More than one Process Pool may exist within a Process.

Required Object	Description
BPM Process Project	A user defined project that allows users to group BPM assets.

Process Pool

Components	Description
Process Swimlane	Allows the grouping of Process Steps by actor.

Process Swimlane

Components	Description
Process Step	Represents an activity in a Process.

Web Application

Components	Description
Web Application Page	Construct that is used to build User Interface pages for CAF web, portlet and task applications.
Portlet	Portlet built using webMethods CAF User Interface technology. Supports JSR 168.
CAF Task Type	webMethods Task Engine human activity task.
CAF Security Role	Java Web Application Security Role.
JDBC Datasource	Connection to a JDBC database.

Portlet

Components	Description
Portlet Preference	Allows customized portlet behavior.
Web Application Page	Construct that is used to build User Interface pages for CAF web, portlet and task applications.

CAF Task Type

Components	Description
CAF Task Rule	webMethods Task Engine Task Assignment or Event execution mechanism.

TN Group

Components	Description
TN Document Type	A description of a document type that is expected in a user's Trading Network.

Business Rules Project

Components	Description
Rule Set	A container for related rule metaphor assets.
Data Model	Defines a set of data elements available to a rule.
Rule Action	Represents some external behavior.

Data Model

Components	Description
Rule Parameter	Represents the connection from a metaphor to a data model.

11 Working with Association Types

An *Association Type* is a logical definition of a relationship. An association type enables you to define a relationship between an asset of one type and an asset of another type and/or to any other object defined in CentraSite. The association type represents the relationship as a forward label or reverse label. An association type can be used by the relationship attribute defined within an asset's profile. For example, if you have an **Association Type** called "Uses", you can define a **Relationship** attribute with this association type "Uses" for a Service asset. You can use that attribute to link the Service asset to other Services in the catalog, to assets of other types (such as XML schemas) and/or to other registry objects (such as Organizations, Policies). For an **Asset Type**, when you define a relationship attribute using this **Association Type** it appears as a attribute on the asset's detail page in the CentraSite Control.

One association type can be used by many asset types. For example, the "Has Parent" association type, which is one of the predefined association types installed with CentraSite, provides parent-child relationship information about an asset and the related to object, and is used by both the Service asset type and the XML schema asset type.

Who Can Create and Manage Association Types?

To create custom association types, you must belong to a role that has the "Manage Asset Types" permission. Besides allowing you to create custom association types, this permission allows you to edit and delete any user-defined association type. Additionally, it allows you to edit certain predefined association types installed by CentraSite. By default, users in the CentraSite Administrator and Asset Type Administrator roles have this permission, although an administrator can grant this permission to other roles.

For more information about permissions, see *About Roles and Permissions* in the document *Users, Groups, Roles and Permissions*.

Adding an Association Type

You use the following procedure to define a new association type.

To define a new association type

1 In CentraSite Control, go to **Administration > Types**. Click the **Association Types** tab.

2 Click **Add Association Type**.

3 In the **Add Association Type** dialog box, specify the following properties:

In this field...	Do the following...
Name	Enter a name for the association type. Be aware that this is the name that will be given to attributes that use this association type. Therefore, the name should be meaningful when used as an attribute name. For example, use an association name such as "Developed By", not "developer association". ■ An association name does not need to be unique within the CentraSite registry. However, to reduce ambiguity, you should avoid giving multiple associations the same name. ■ An association name can contain any character (including spaces).
Forward Label	Specify the relationship of the source asset (the one in which the "Relationship" attribute resides) to one or more specified targets. If you are not specifying a name for the forward label, then CentraSite will treat the association type name as the forward label.
Reverse Label	*Optional*. Specify the relationship of the specified targets to the source asset.

4 When you finish setting the association type's properties, click **OK**.

Editing the Properties of an Association Type

When you attempt to modify an association type, keep the following points in mind:

- You can modify the association type's name at any time.
- You can modify an association type for relationship properties *only if there is no relationship attribute defined with it in the catalog*. After a association type has been assigned to an attribute, it can no longer be edited.

To modify an association type

1 In CentraSite Control, go to **Administration > Types**.

2 In the **Association Types** tab, select the association type's link that you want to modify.

3 Examine or modify the properties on the **Edit Association Type** dialog box as required.

4 Click **OK**.

Deleting an Association Type

When you attempt to delete an association type, keep the following points in mind:

- You cannot delete the association types that CentraSite provides out-of-the-box (not even if you belong to a role with the "Manage Asset Types" permission).
- You can delete an association type *only if there is no relationship attribute defined with it in the catalog*. After a association type has been assigned to a relationship attribute, it can no longer be deleted.
- You cannot delete the stand-alone association type that is in use for defining an association in the catalog.

To delete an association type

1 In the CentraSite Control, go to **Administration > Types**.

2 Go to the **Association Types** tab.

3 Enable the checkbox next to the name of an association type that you want to delete.

4 Click **Delete**.

When you are prompted to confirm the delete operation, click **OK**.

You can delete multiple association types in a single step. The rules described above for deleting a single association type apply also when deleting multiple association types.

Important: If you have selected several association types, you can use the **Delete** button to delete the types. However, you are not allowed to delete types for which you do not have the required permission.

To delete multiple association types in a single operation

1 In CentraSite Control, go to **Administration > Types**.

2 Go to the **Association Types** tab.

3 Mark the checkboxes of the types that you want to delete.

4 From the **Actions** menu, choose **Delete**.

When you are prompted to confirm the delete operation, click **OK**.